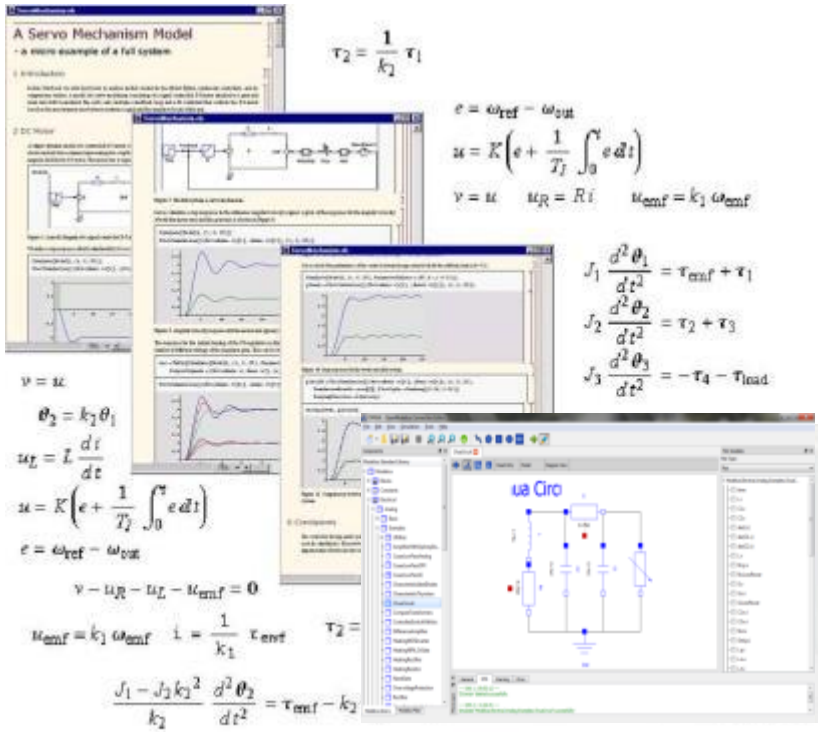
Introduction to Object-Oriented Modeling and Simulation with Modelica and OpenModelica



Tutorial, Version Feb 03, 2015

Peter Fritzson

Linköping University, peter.fritzson@liu.se
Director of the Open Source Modelica Consortium
Vice Chairman of Modelica Association

Mahder Gebremedhin, PhD student
mahder.gebremedhin@liu.se

Slides

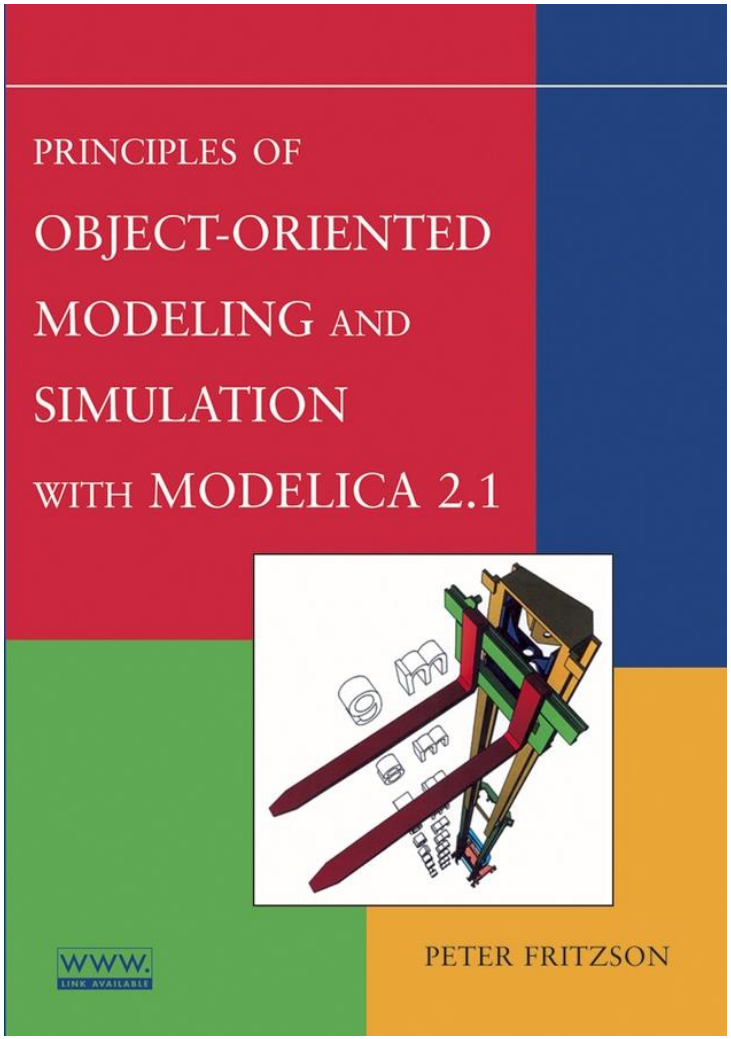
Based on book and lecture notes by Peter Fritzson
Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus
Contributions 2006-2008 by Adrian Pop and Peter Fritzson
Contributions 2009 by David Broman, Peter Fritzson, Jan Brugård, and Mohsen Torabzadeh-Tari
Contributions 2010 by Peter Fritzson
Contributions 2011 by Peter F., Mohsen T,. Adeel Asghar,
Contributions 2012, 2013, 2014, 2015 by Peter Fritzson, Lena Buffoni, and Mahder Gebremedhin

2015-02-03

Linköpings universitet

MODELICA

Tutorial Based on Book, 2004
Download OpenModelica Software



Peter Fritzson

Principles of Object Oriented Modeling and Simulation with Modelica 2.1

Wiley-IEEE Press, 2004, 940 pages

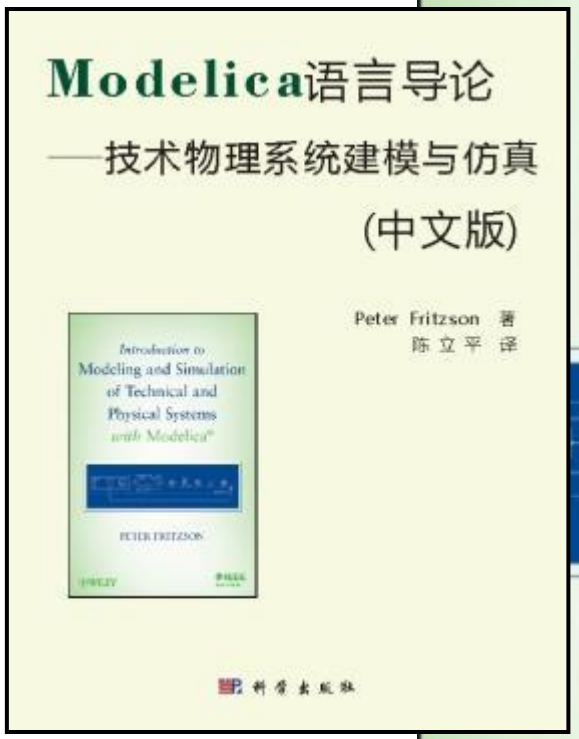
- OpenModelica
  - www.openmodelica.org
- Modelica Association
  - www.modelica.org

Copyright © Open Source Modelica Consortium

Introductory Modelica Book

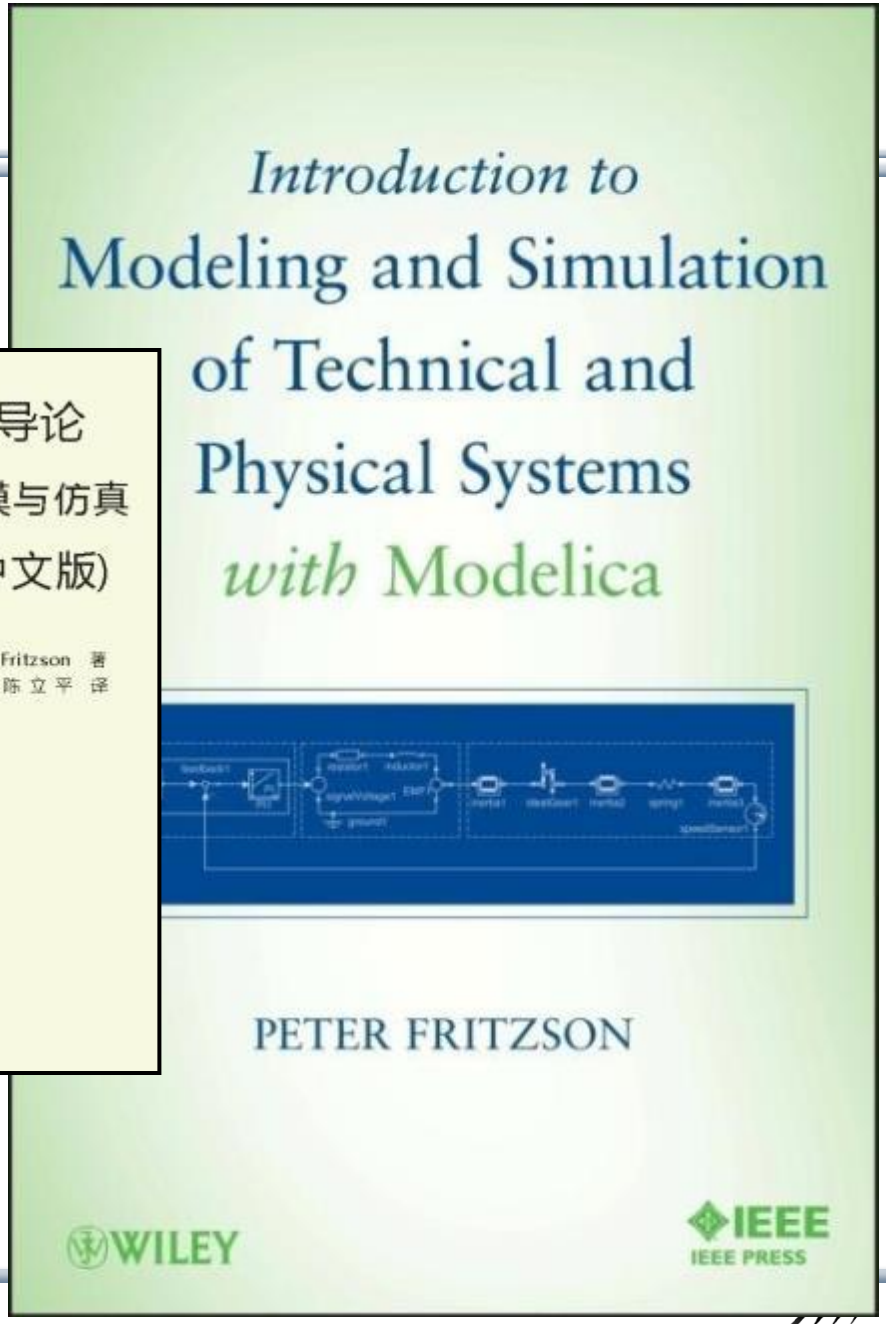
September 2011
232 pages

2014 –Translations available in **Chinese**, **Japanese**, **Spanish**



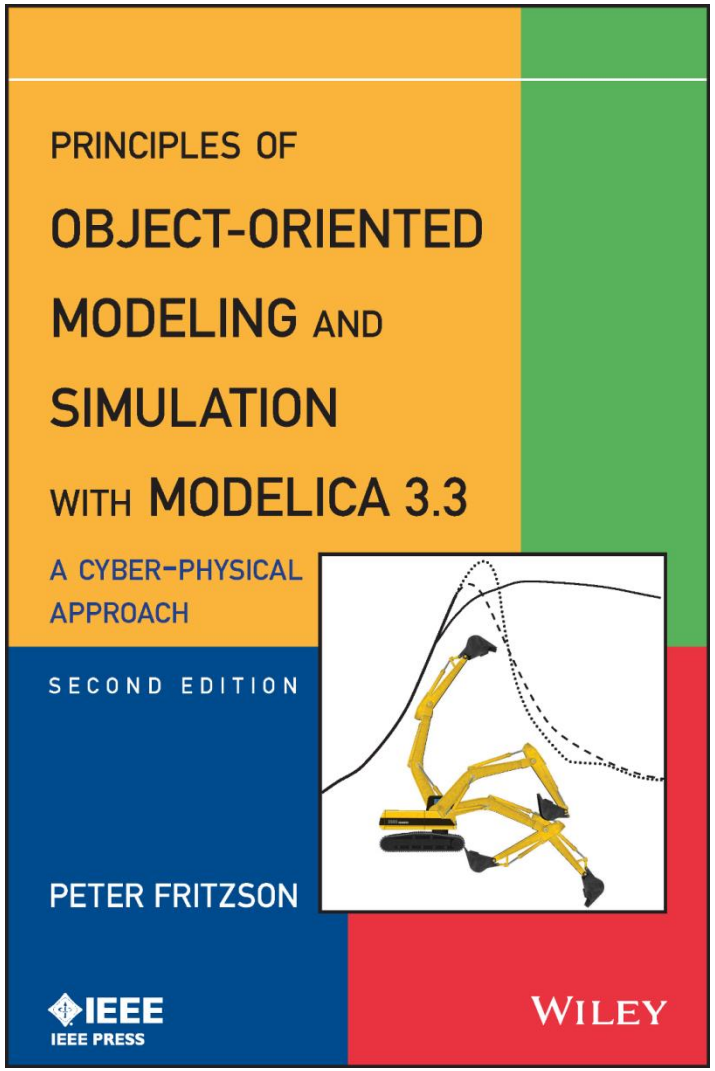
Wiley
IEEE Press

For Introductory Short Courses on Object Oriented Mathematical Modeling



Copyright © Open Source Modelica Consortium

New Big Modelica Book, 2014



Peter Fritzson

Principles of Object Oriented Modeling and Simulation with Modelica 3.3

A Cyber-Physical Approach

Can be ordered from Wiley or Amazon

Wiley-IEEE Press, 2014, 1250 pages

- OpenModelica
  - www.openmodelica.org
- Modelica Association
  - www.modelica.org

Copyright © Open Source Modelica Consortium

Acknowledgements, Usage, Copyrights

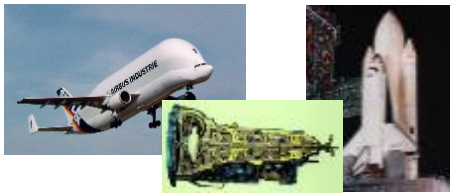
- If you want to use the Powerpoint version of these slides in your own course, send an email to: peter.fritzson@ida.liu.se
- Thanks to Emma Larsdotter Nilsson, Peter Bunus, David Broman, Jan Brugård, Mohsen-Torabzadeh-Tari, Adeel Asghar, Lena Buffoni, for contributions to these slides.
- Most examples and figures in this tutorial are adapted with permission from Peter Fritzson's book "Principles of Object Oriented Modeling and Simulation with Modelica 2.1", copyright Wiley-IEEE Press
- Some examples and figures reproduced with permission from Modelica Association, Martin Otter, Hilding Elmqvist, Wolfram MathCore, Siemens
- Modelica Association: www.modelica.org
- OpenModelica: www.openmodelica.org

Copyright © Open Source Modelica Consortium

Outline

Part I

Introduction to Modelica and a demo example

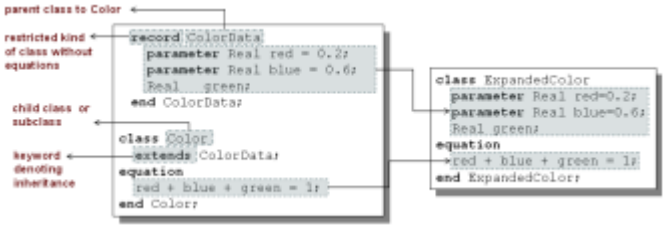


Part II

Modelica environments

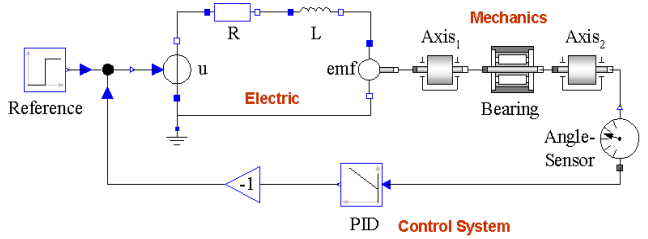
Part III

Modelica language concepts and textual modeling



Part IV

Graphical modeling and the Modelica standard library



Copyright © Open Source Modelica Consortium

Detailed Schedule (afternoon version)

14:00 - Introduction to Modeling and Simulation
- Start installation of **OpenModelica** including **OMEdit** graphic editor

14:10 - Modelica – The Next Generation Modeling Language

14:25 - *Exercises Part I (15 minutes)*
- Short hands-on exercise on graphical modeling using **OMEdit**– RL Circuit

14:50 – Part II: Modelica Environments and the OpenModelica Environment

15:10 – Part III: Modelica Textual Modeling

15:15 - *Exercises Part IIIa (30 minutes)*
- Hands-on exercises on textual modeling using the **OpenModelica** environment

15:45 – Coffee Break

16:00 - Modelica Discrete Events and Hybrid Properties

16:15 - *Exercises Part IIIb (10 minutes)*
- Hands-on exercises on textual modeling using the **OpenModelica** environment

16:25 – Part IV: Components, Connectors and Connections
- Modelica Libraries

16:45 - Graphical Modeling using OpenModelica

17:00 - *Exercises Part IV (30 minutes) – DCMotor etc.*
- Hands-on exercises on graphical modeling using **OpenModelica**

Copyright © Open Source Modelica Consortium

Software Installation - Windows

- Start the software installation
- Install OpenModelica-1.9.2beta.exe from the USB Stick

Copyright © Open Source Modelica Consortium

Software Installation – Linux (requires internet connection)

- Go to https://openmodelica.org/index.php/download/download-linux and follow the instructions.

Copyright © Open Source Modelica Consortium

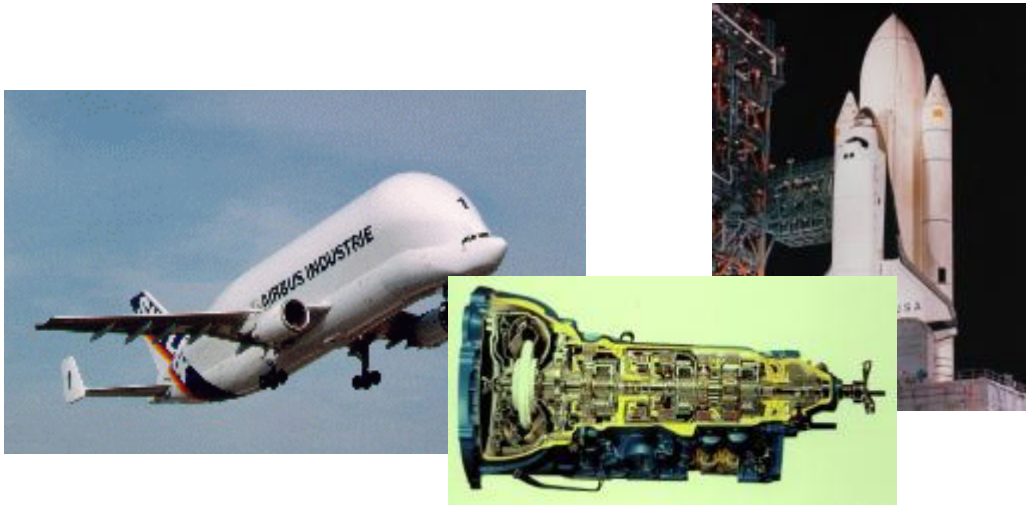
Software Installation – MAC (requires internet connection)

- Go to https://openmodelica.org/index.php/download/download-mac and follow the instructions or follow the instructions written below.
- The installation uses MacPorts. After setting up a MacPorts installation, run the following commands on the terminal (as root):
  - *echo rsync://build.openmodelica.org/macports/ >> /opt/local/etc/macports/sources.conf # assuming you installed into /opt/local*
  - *port selfupdate*
  - *port install openmodelica-devel*

Copyright © Open Source Modelica Consortium

Part I

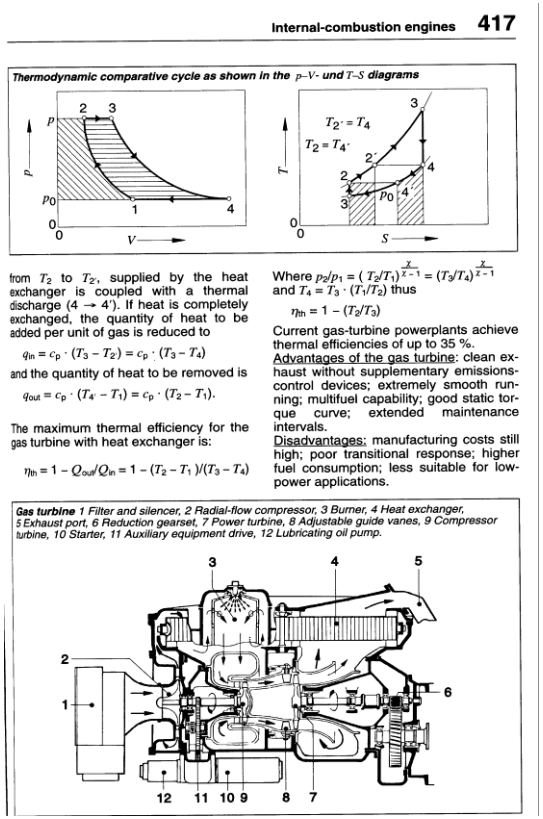
Introduction to Modelica and
a demo example



Copyright © Open Source Modelica Consortium

Modelica Background: Stored Knowledge

Model knowledge is stored in books and human minds which computers cannot access



Internal-combustion engines 417

Thermodynamic comparative cycle as shown in the p–V- und T–S diagrams

from T_2 to $T_{2'}$, supplied by the heat exchanger is coupled with a thermal discharge (4 → 4'). If heat is completely exchanged, the quantity of heat to be added per unit of gas is reduced to

$$q_{in} = c_p \cdot (T_3 - T_{2'}) = c_p \cdot (T_3 - T_4)$$

and the quantity of heat to be removed is

$$q_{out} = c_p \cdot (T_{4'} - T_1) = c_p \cdot (T_2 - T_1).$$

The maximum thermal efficiency for the gas turbine with heat exchanger is:

$$\eta_{th} = 1 - Q_{out}/Q_{in} = 1 - (T_2 - T_1)/(T_3 - T_4)$$

Where $p_2/p_1 = (T_2/T_1)^{\frac{\chi}{\chi-1}} = (T_3/T_4)^{\frac{\chi}{\chi-1}}$ and $T_4 = T_3 \cdot (T_1/T_2)$ thus

$$\eta_{th} = 1 - (T_2/T_3)$$

Current gas-turbine powerplants achieve thermal efficiencies of up to 35 %.

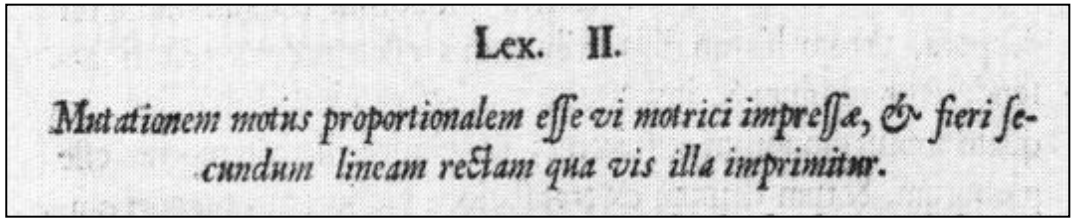
Advantages of the gas turbine: clean exhaust without supplementary emissions-control devices; extremely smooth running; multifuel capability; good static torque curve; extended maintenance intervals.

Disadvantages: manufacturing costs still high; poor transitional response; higher fuel consumption; less suitable for low-power applications.

Gas turbine 1 Filter and silencer, 2 Radial-flow compressor, 3 Burner, 4 Heat exchanger, 5 Exhaust port, 6 Reduction gearset, 7 Power turbine, 8 Adjustable guide vanes, 9 Compressor turbine, 10 Starter, 11 Auxiliary equipment drive, 12 Lubricating oil pump.

"The change of motion is proportional to the motive force impressed"

– Newton



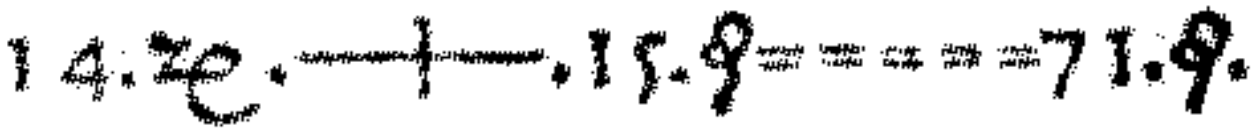
Lex. II.

Mutationem motus proportionalem esse vi motrici impressæ, & fieri secundum lineam rectam qua vis illa imprimitur.

Copyright © Open Source Modelica Consortium

Modelica Background: The Form – Equations

- Equations were used in the third millennium B.C.
- Equality sign was introduced by Robert Recorde in 1557



Newton still wrote text (Principia, vol. 1, 1686)

"*The change of motion is proportional to the motive force impressed*"

CSSL (1967) introduced a special form of "equation":

```
variable = expression
v = INTEG(F)/m
```

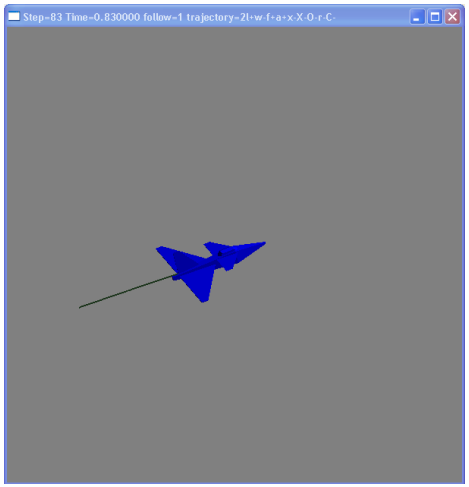
Programming languages usually do not allow equations!

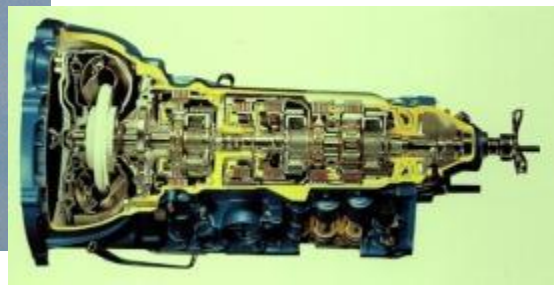
Copyright © Open Source Modelica Consortium

What is Modelica?

A language for modeling of complex cyber-physical systems

- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology

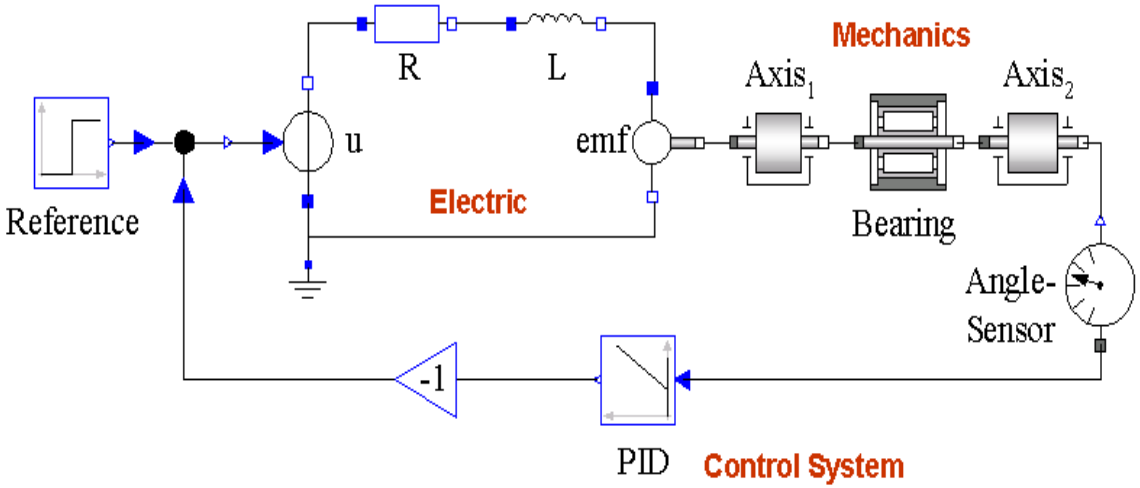




Copyright © Open Source Modelica Consortium

What is Modelica?

A language for modeling of complex cyber-physical systems



Primary designed for **simulation**, but there are also other usages of models, e.g. optimization.

Copyright © Open Source Modelica Consortium

What is Modelica?

A language for modeling of complex cyber-physical systems

i.e., Modelica is **not** a tool

Free, open language specification:

Available at: www.modelica.org

Developed and standardized by Modelica Association

There exist several free and commercial tools, for example:

- **OpenModelica from OSMC**
- Dymola from Dassault systems
- Wolfram System Modeler fr Wolfram MathCore
- SimulationX from ITI
- MapleSim from MapleSoft
- AMESIM from LMS
- JModelica.org from Modelon
- MWORKS from Tongyang Sw & Control
- IDA Simulation Env, from Equa
- CyModelica Modeling tool, ESI Group

Copyright © Open Source Modelica Consortium

Modelica – The Next Generation Modeling Language

Declarative language

Equations and mathematical functions allow acausal modeling, high level specification, increased correctness

Multi-domain modeling

Combine electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc...

Everything is a class

Strongly typed object-oriented language with a general class concept, Java & MATLAB-like syntax

Visual component programming

Hierarchical system architecture capabilities

Efficient, non-proprietary

Efficiency comparable to C; advanced equation compilation, e.g. 300 000 equations, ~150 000 lines on standard PC

Copyright © Open Source Modelica Consortium

Modelica Acausal Modeling

What is *acausal* modeling/design?

Why does it increase *reuse*?

> The acausality makes Modelica library classes *more reusable* than traditional classes containing assignment statements where the input-output causality is fixed.

Example: a resistor *equation*:

R*i = v;

can be used in three ways:

i := v/R;

v := R*i;

R := v/i;

Copyright © Open Source Modelica Consortium

What is Special about Modelica?

- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation

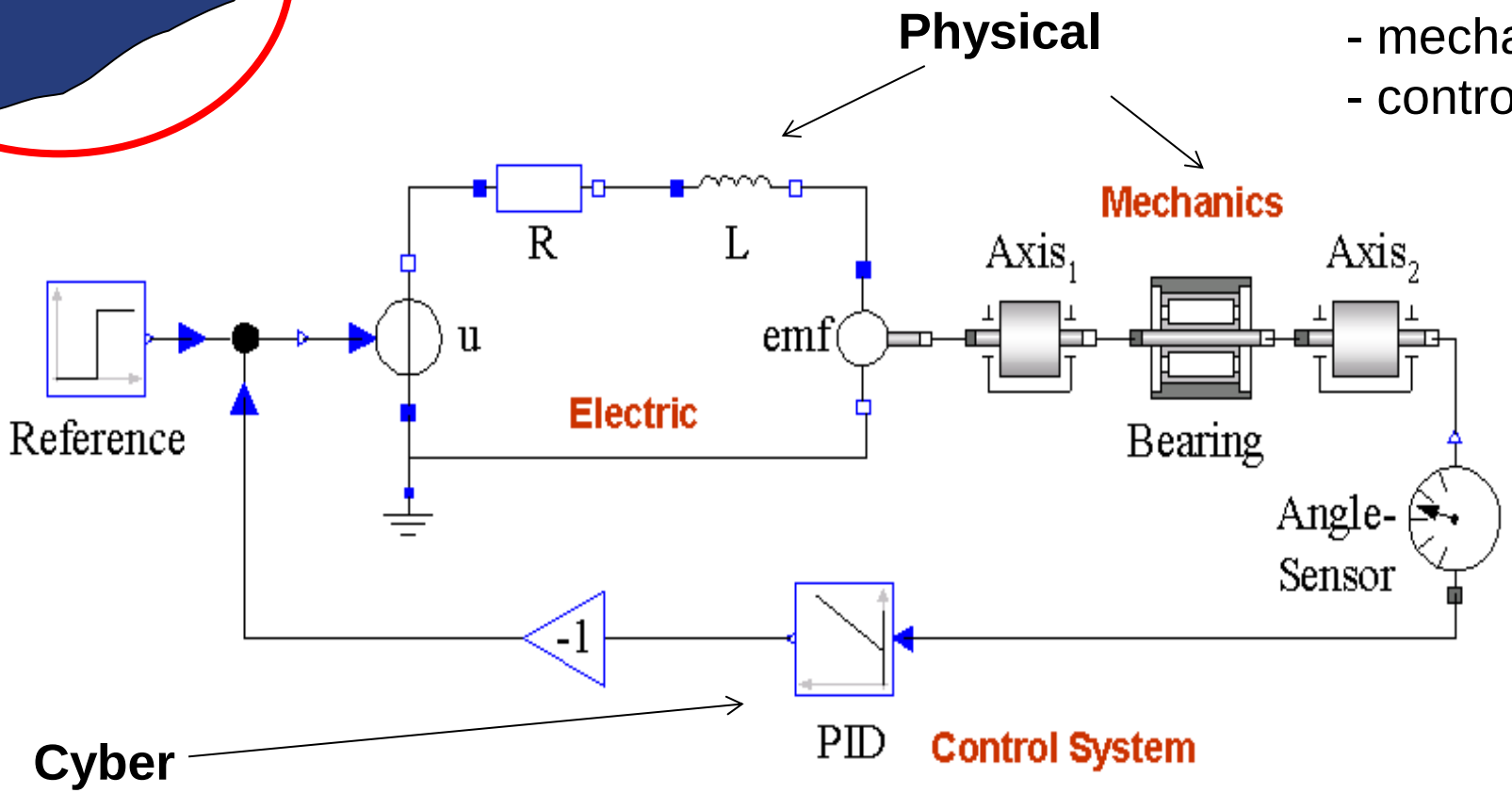
Copyright © Open Source Modelica Consortium

What is Special about Modelica?

Multi-Domain Modeling

Cyber-Physical Modeling

3 domains
- electric
- mechanics
- control



Copyright © Open Source Modelica Consortium

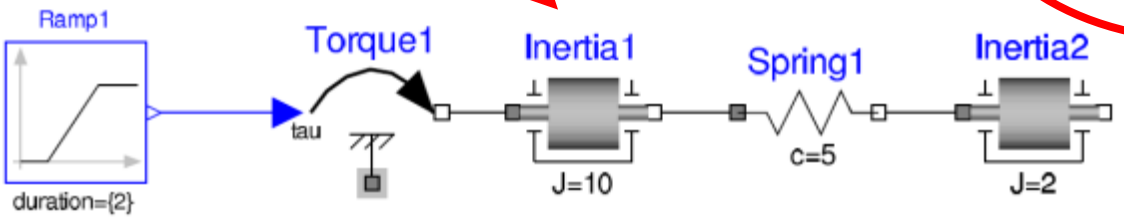
What is Special about Modelica?

Multi-Domain Modeling

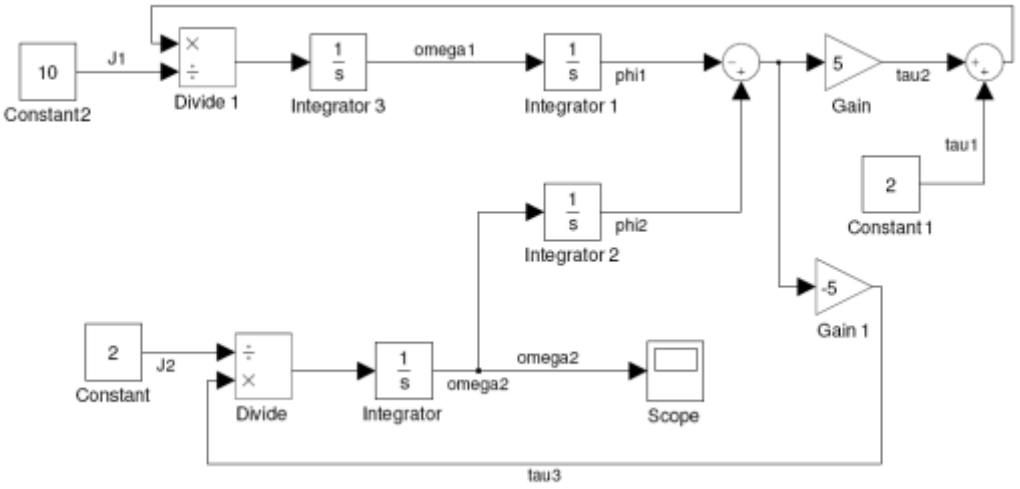
Visual Acausal Hierarchical Component Modeling

Keeps the physical structure

Acausal model (Modelica)



Causal block-based model (Simulink)



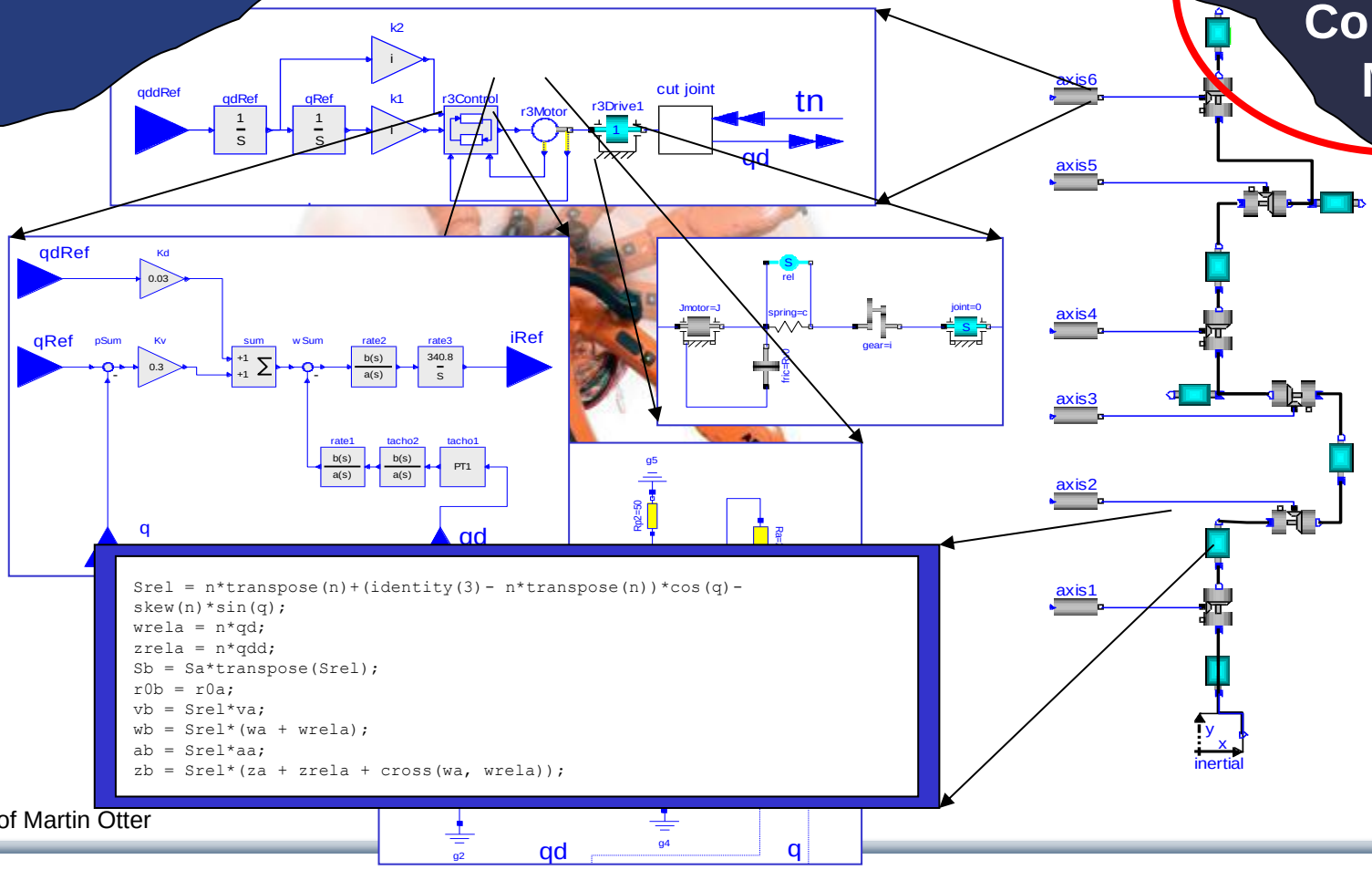
Copyright © Open Source Modelica Consortium

What is Special about Modelica?

Multi-Domain Modeling

Hierarchical system modeling

Visual Acausal Hierarchical Component Modeling



Courtesy of Martin Otter

Copyright © Open Source Modelica Consortium

What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Hierarchical Component Modeling

A textual *class-based* language
OO primary used for as a structuring concept

Behaviour described declaratively using

- Differential algebraic equations (DAE) (continuous-time)
- Event triggers (discrete-time)

Typed Declarative Equation-based Textual Language

Variable declarations

```
class VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x";
  Real y(start = 1)  "y coordinate";
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = -x + lambda*(1 - x*x)*y;
end VanDerPol;
```

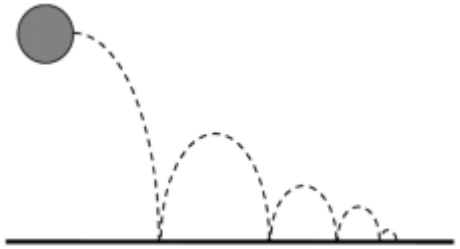
Differential equations

Copyright © Open Source Modelica Consortium

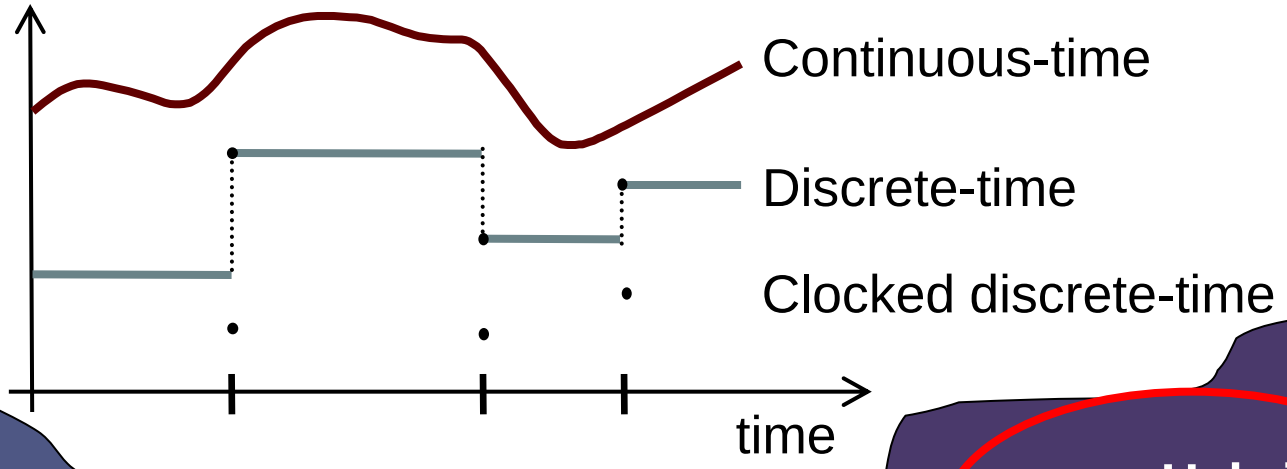
What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Component Modeling



Hybrid modeling =
continuous-time + discrete-time modeling



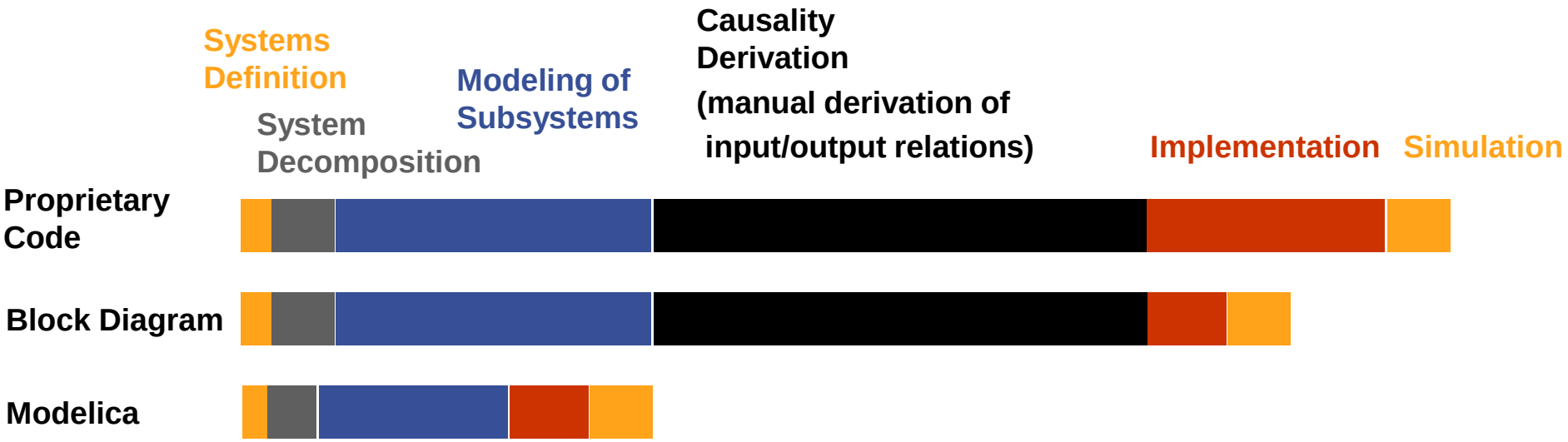
Typed
Declarative
Equation-based
Textual Language

Hybrid Modeling

Copyright © Open Source Modelica Consortium

Modelica – Faster Development, Lower Maintenance than with Traditional Tools

Block Diagram (e.g. Simulink, ...) or
Proprietary Code (e.g. Ada, Fortran, C,...)
vs Modelica

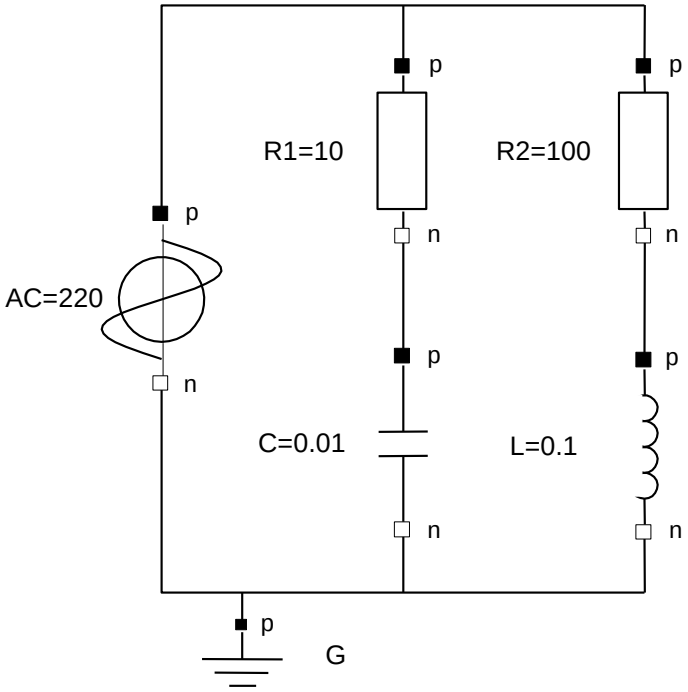


Copyright © Open Source Modelica Consortium

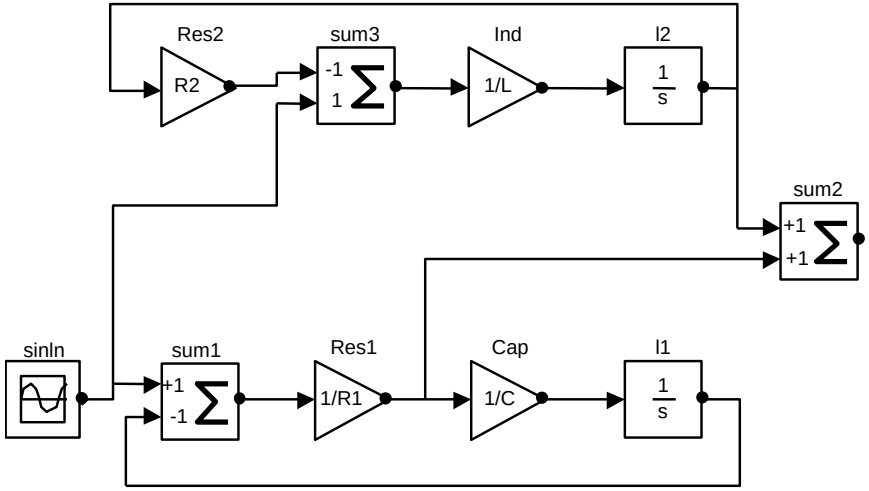
Modelica vs Simulink Block Oriented Modeling Simple Electrical Model

Modelica:
Physical model – easy to understand

Keeps the physical structure

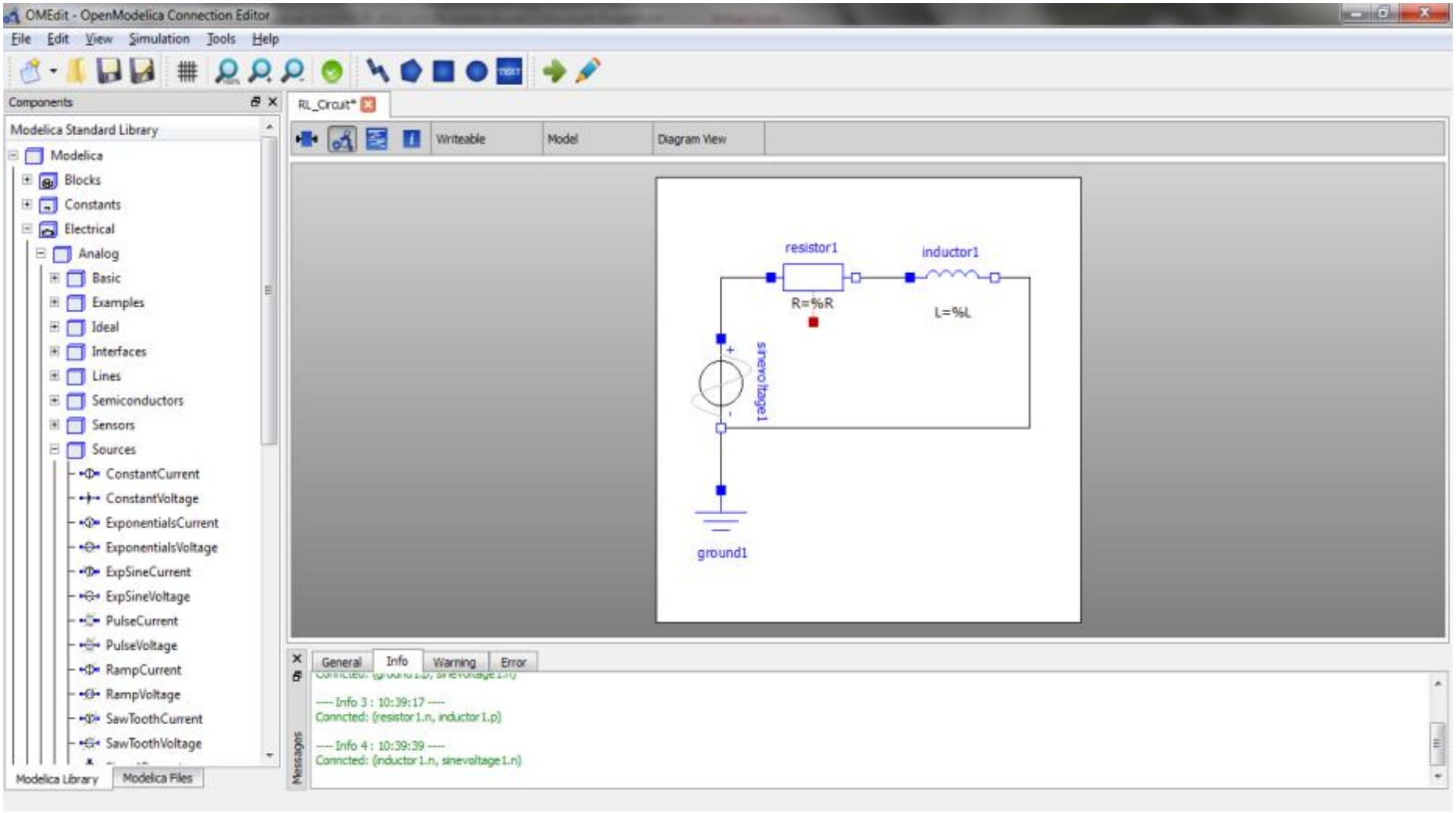


Simulink:
Signal-flow model – hard to understand



Copyright © Open Source Modelica Consortium

Graphical Modeling - Using Drag and Drop Composition

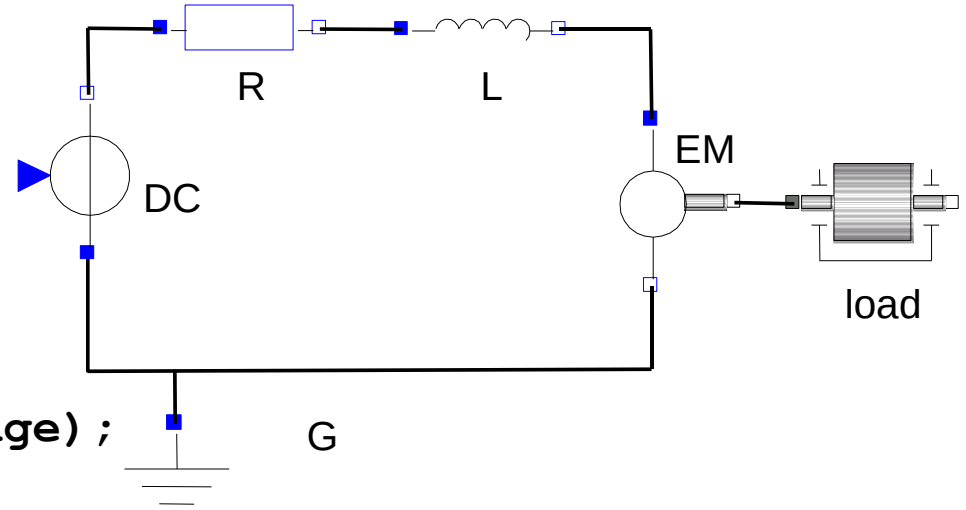


Copyright © Open Source Modelica Consortium

Multi-Domain (Electro-Mechanical) Modelica Model

- A DC motor can be thought of as an electrical circuit which also contains an electromechanical component

```
model DCMotor
   Resistor R(R=100);
   Inductor L(L=100);
   VsourceDC DC(f=10);
   Ground G;
   ElectroMechanicalElement EM(k=10,J=10, b=2);
   Inertia load;
equation
   connect(DC.p,R.n);
   connect(R.p,L.n);
   connect(L.p, EM.n);
   connect(EM.p, DC.n);
   connect(DC.n,G.p);
   connect(EM.flange,load.flange);
end DCMotor
```



Copyright © Open Source Modelica Consortium

Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

```
0 == DC.p.i + R.n.i
DC.p.v == R.n.v

0 == R.p.i + L.n.i
R.p.v == L.n.v

0 == L.p.i + EM.n.i
L.p.v == EM.n.v

0 == EM.p.i + DC.n.i
EM.p.v == DC.n.v

0 == DC.n.i + G.p.i
DC.n.v == G.p.v
```

```
EM.u == EM.p.v - EM.n.v
0 == EM.p.i + EM.n.i
EM.i == EM.p.i
EM.u == EM.k * EM.ω
EM.i == EM.M / EM.k
EM.J * EM.ω == EM.M - EM.b * EM.ω

DC.u == DC.p.v - DC.n.v
0 == DC.p.i + DC.n.i
DC.i == DC.p.i
DC.u == DC.Amp * Sin[2 π DC.f * t]
```

```
R.u == R.p.v - R.n.v
0 == R.p.i + R.n.i
R.i == R.p.i
R.u == R.R * R.i

L.u == L.p.v - L.n.v
0 == L.p.i + L.n.i
L.i == L.p.i
L.u == L.L * L.i'
```

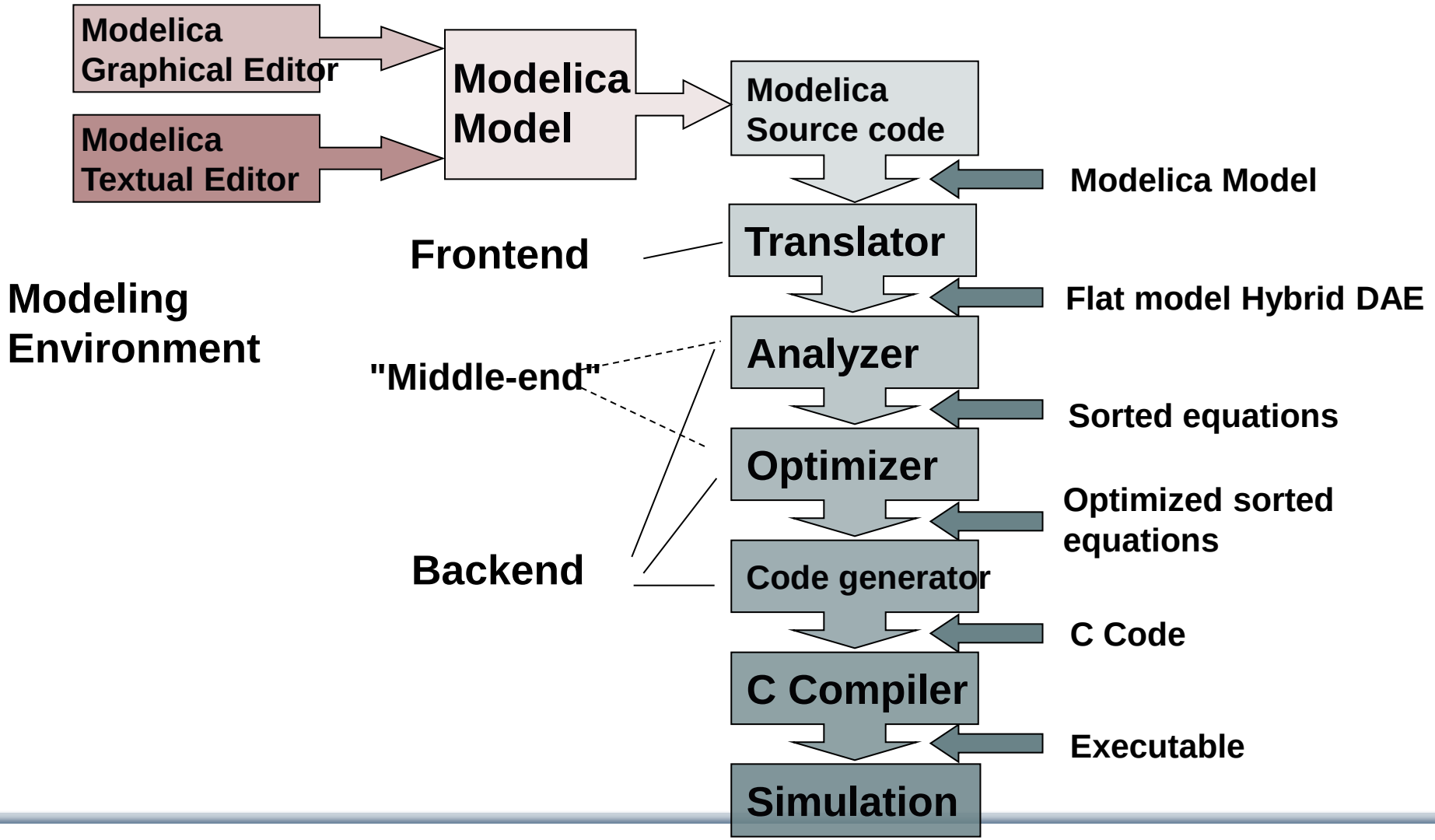
(`load` component not included)

Automatic transformation to ODE or DAE for simulation:

$$\frac{dx}{dt} == f[x, u, t] \qquad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

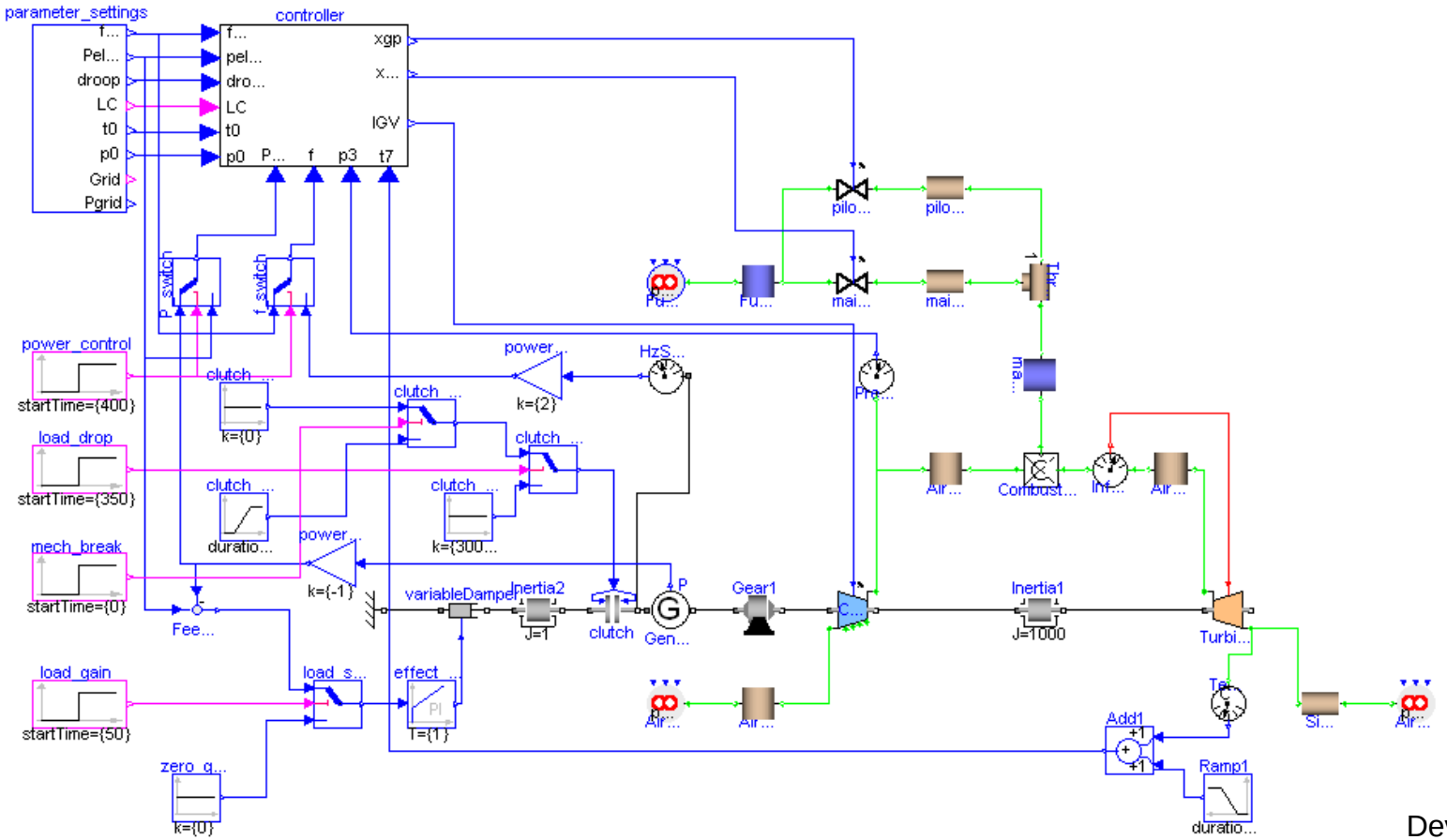
Copyright © Open Source Modelica Consortium

Model Translation Process to Hybrid DAE to Code



Copyright © Open Source Modelica Consortium

Modelica in Power Generation
GTX Gas Turbine Power Cutoff Mechanism

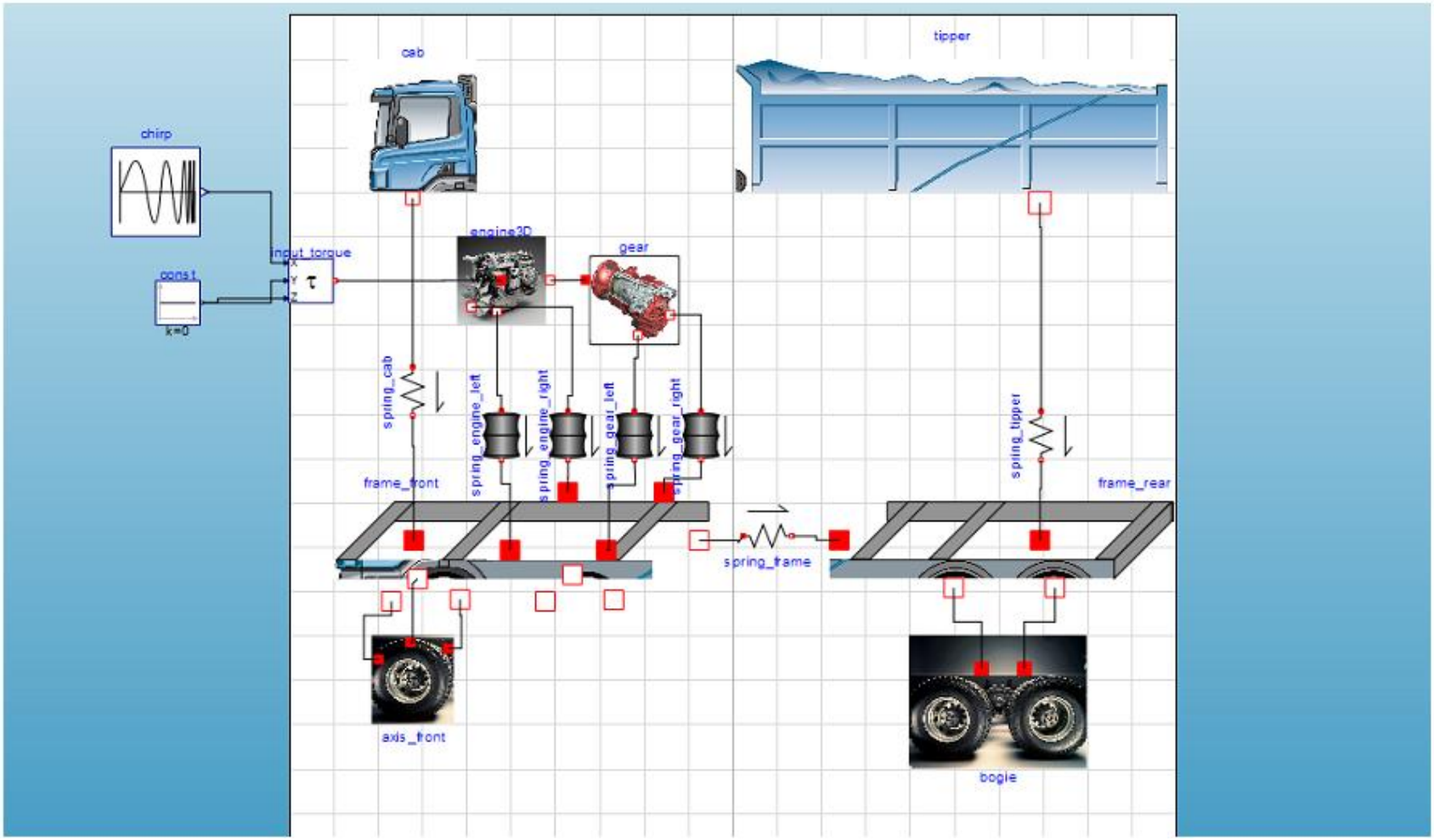


Developed by MathCore for Siemens

Courtesy of Siemens Industrial Turbomachinery AB

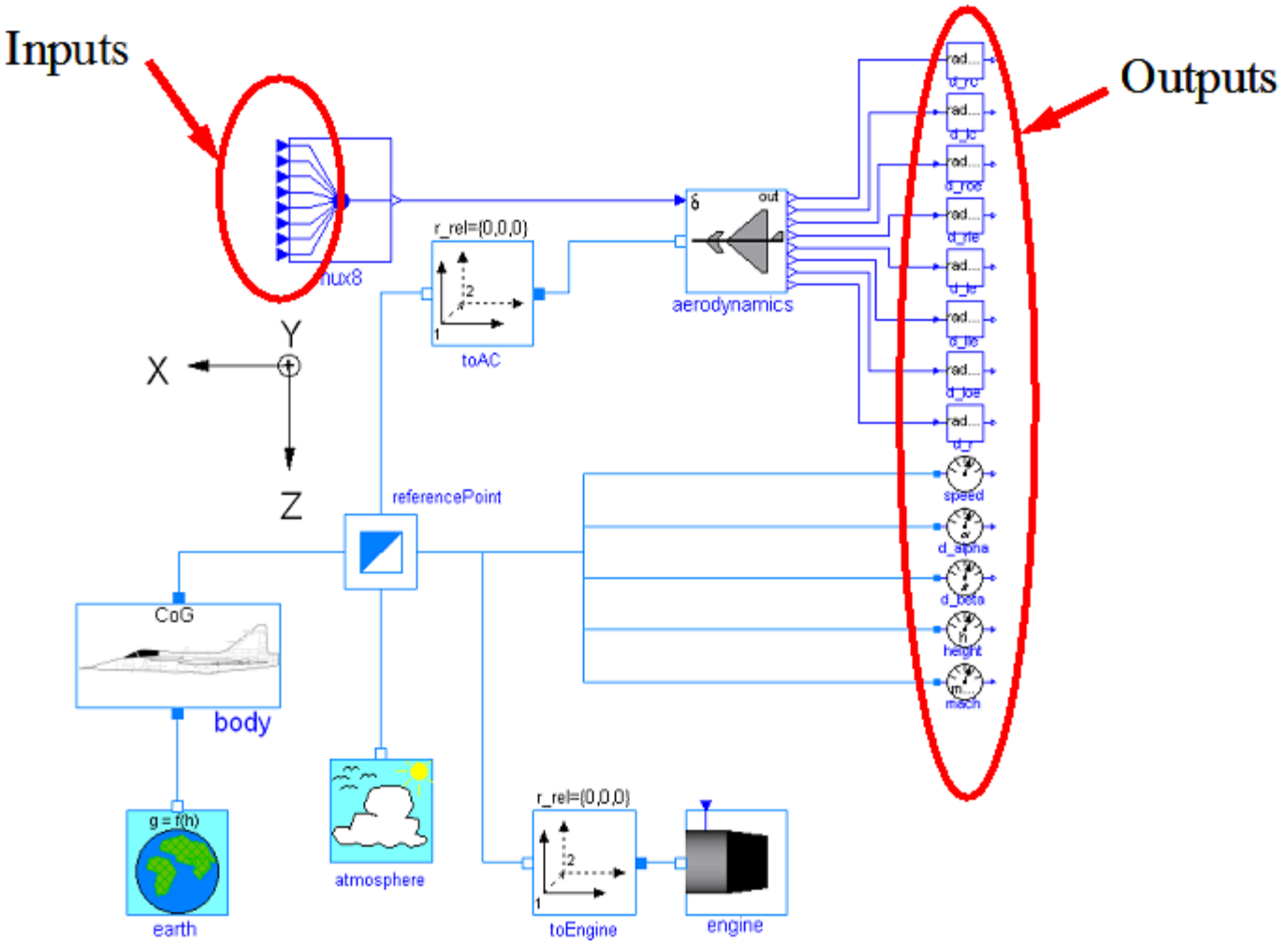
Copyright © Open Source Modelica Consortium

Modelica in Automotive Industry



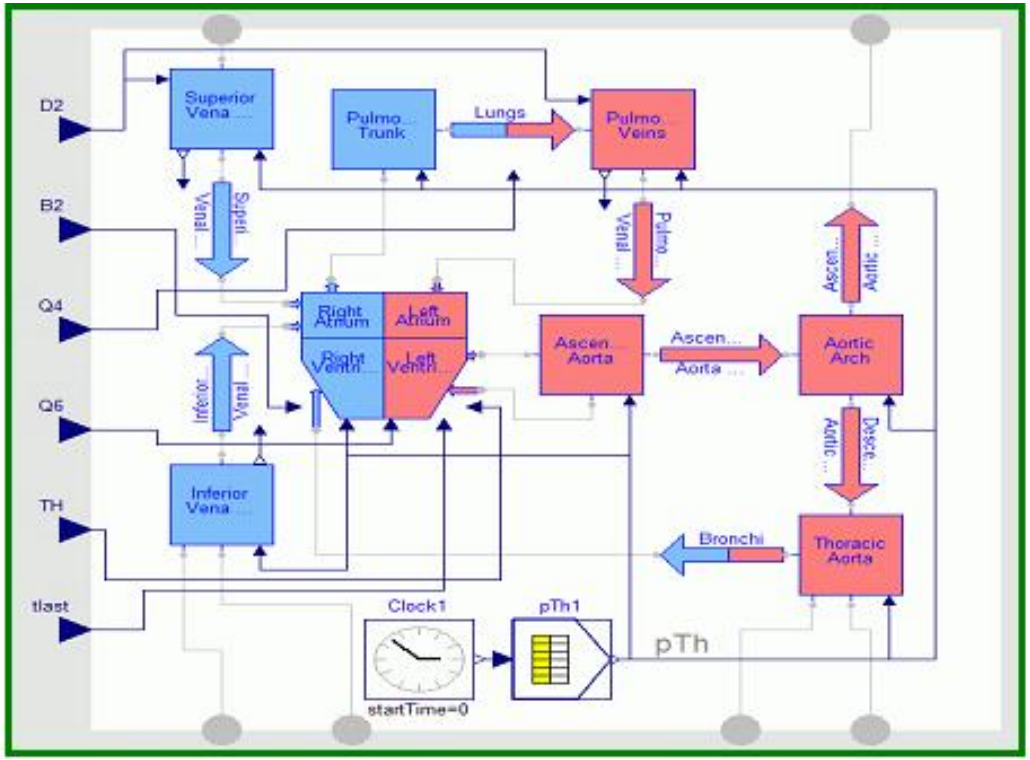
Copyright © Open Source Modelica Consortium

Modelica in Avionics



Copyright © Open Source Modelica Consortium

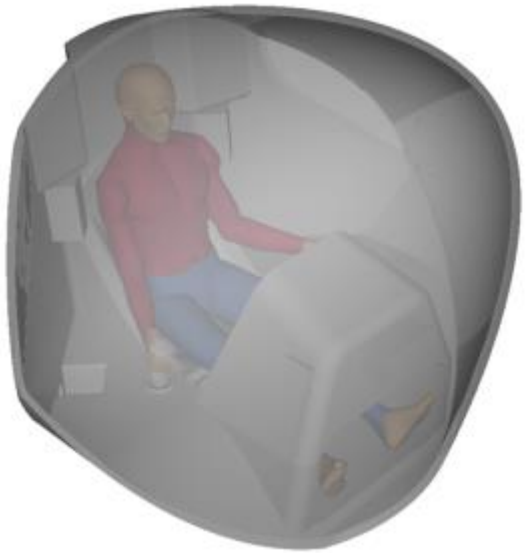
Modelica in Biomechanics



Copyright © Open Source Modelica Consortium

Application of Modelica in Robotics Models Real-time Training Simulator for Flight, Driving

- Using Modelica models generating real-time code
- Different simulation environments (e.g. Flight, Car Driving, Helicopter)
- Developed at DLR Munich, Germany
- Dymola Modelica tool

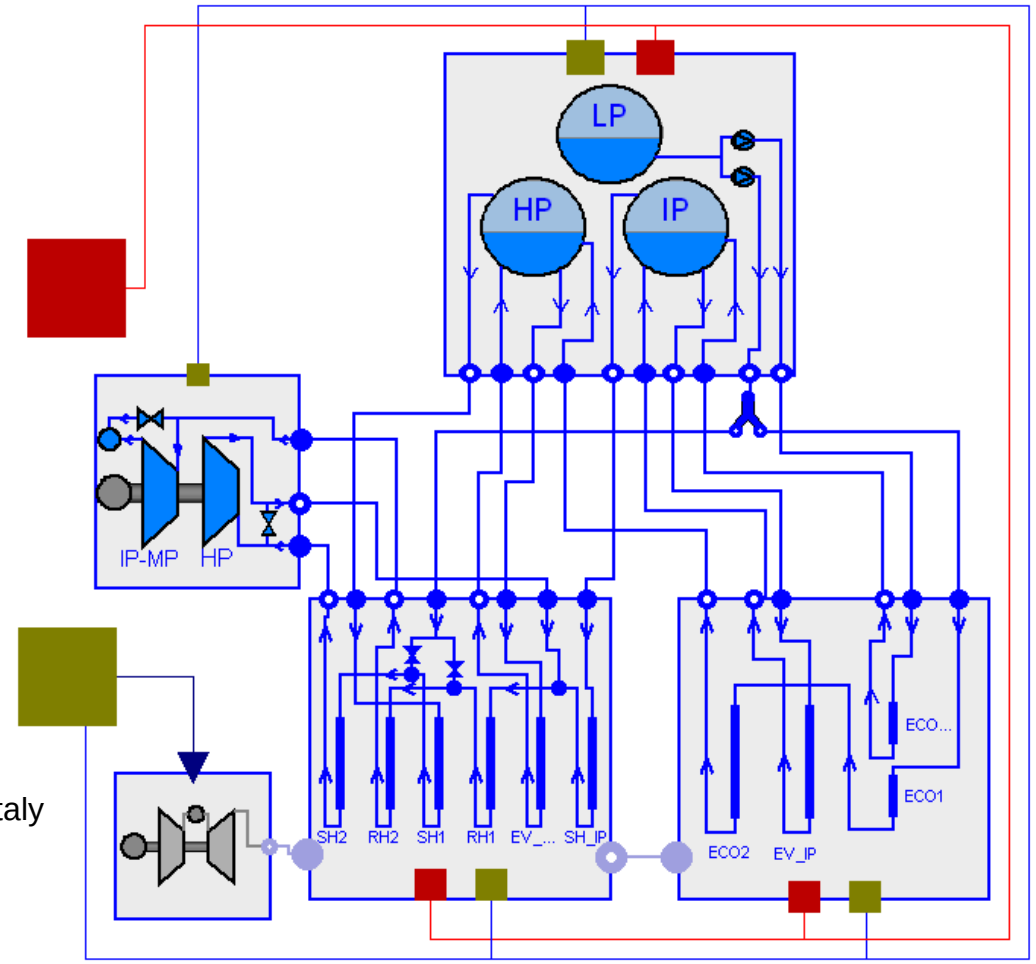


Courtesy of Martin Otter, DLR, Oberphaffenhofen, Germany

Copyright © Open Source Modelica Consortium

Combined-Cycle Power Plant
Plant model – system level

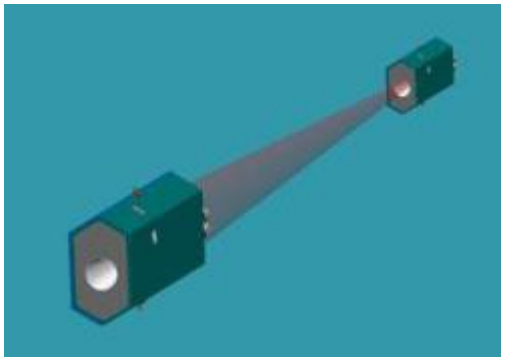
- GT unit, ST unit, Drum boilers unit and HRSG units, connected by thermo-fluid ports and by signal buses
- Low-temperature parts (condenser, feedwater system, LP circuits) are represented by trivial boundary conditions.
- GT model: simple law relating the electrical load request with the exhaust gas temperature and flow rate.



Courtesy Francesco Casella, Politecnico di Milano – Italy and Francesco Pretolani, CESI SpA - Italy

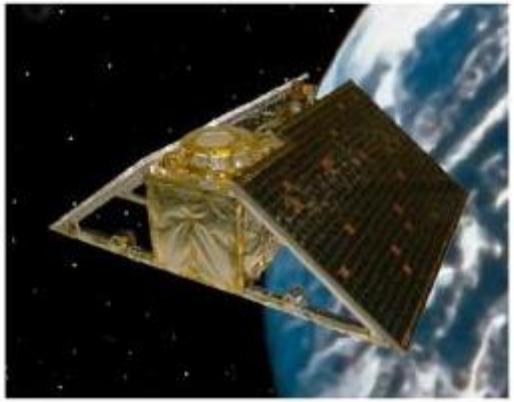
Copyright © Open Source Modelica Consortium

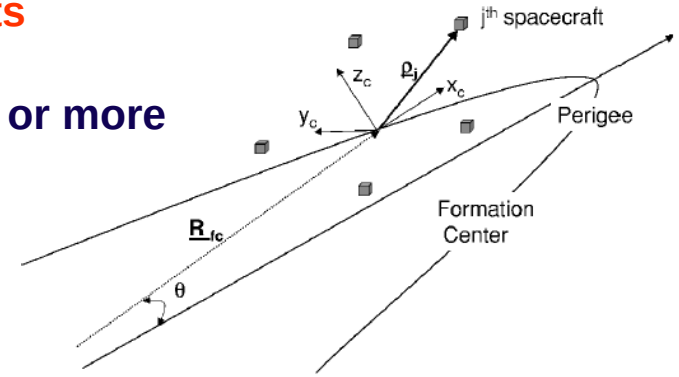
Modelica Spacecraft Dynamics Library



Formation flying on elliptical orbits

Control the relative motion of two or more spacecraft





Attitude control for satellites using magnetic coils as actuators

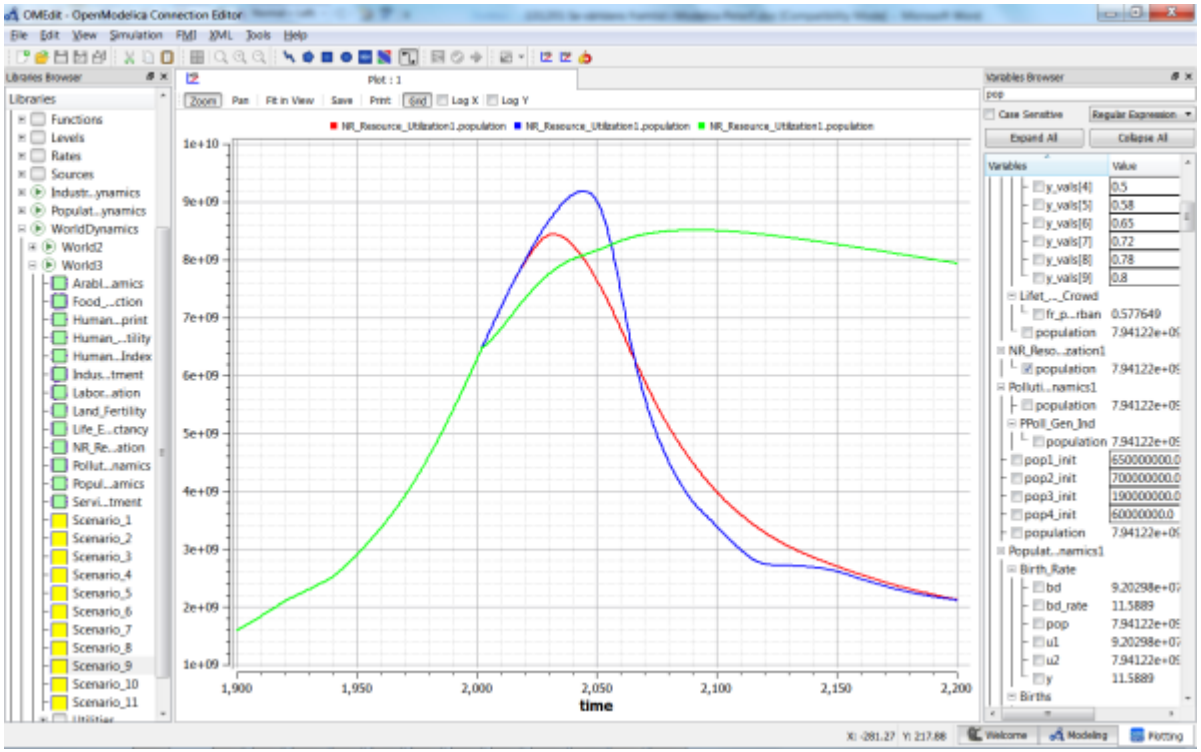
Torque generation mechanism: interaction between coils and geomagnetic field

Courtesy of Francesco Casella, Politecnico di Milano, Italy

Copyright © Open Source Modelica Consortium

System Dynamics – World Society Simulation

Limits to Material Growth; Population, Energy and Material flows



Left. World3 simulation with OpenModelica

- 2 collapse scenarios (close to current developments)
- 1 sustainable scenario (green).

CO_2 Emissions per person**:**

- USA 17 ton/yr
- Sweden 7 ton/yr
- India 1.4 ton/yr
- Bangladesh 0.3 ton/yr

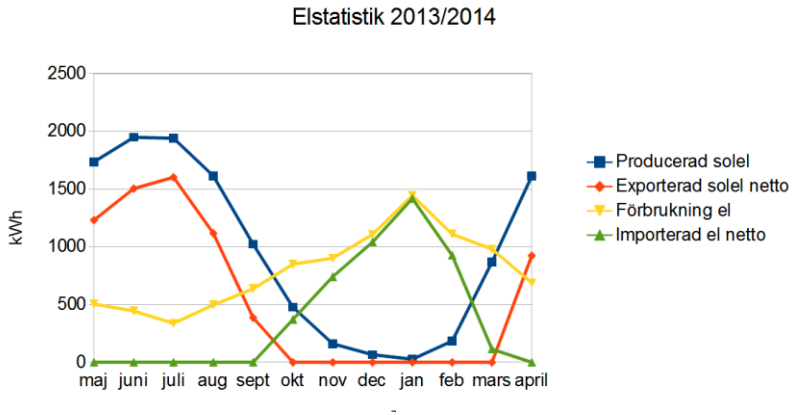
- System Dynamics Modelica library by Francois Cellier (ETH), et al in OM distribution.
- Warming converts many agriculture areas to deserts (USA, Europe, India, Amazonas)
- Ecological breakdown around 2080-2100, drastic reduction of world population
- To **avoid** this: Need for massive investments in sustainable technology and renewable energy sources

Copyright © Open Source Modelica Consortium

What Can You Do?
Need Global Sustainability Mass Movement

- Book: Current catastrophic scenarios: ***Mark Lynas*: ”6 Degrees”**
  Book: How to address the problems: ***Tim Jackson*** **”Prosperity without Growth”**
- Promote sustainable lifestyle and technology
- Install electric solar PV panels
- Buy shares in cooperative wind power

20 sqm solar panels on garage roof, Nov 2012
Generated 2700 W at noon March 10, 2013



Expanded to 93 sqm, 12 kW, March 2013
House produced 11600 kwh, used 9500 kwh
Avoids 10 ton CO_2 emission per year

Copyright © Open Source Modelica Consortium

Example Electric Cars

Can be charged by electricity from own solar panels

Renault ZOE; 5 seat; Range:

- **EU-drive cycle 210 km**
- **Realistic Swedish drive cycle:**
- **Summer: 165 km**
- **Winter: 100 – 110 km**

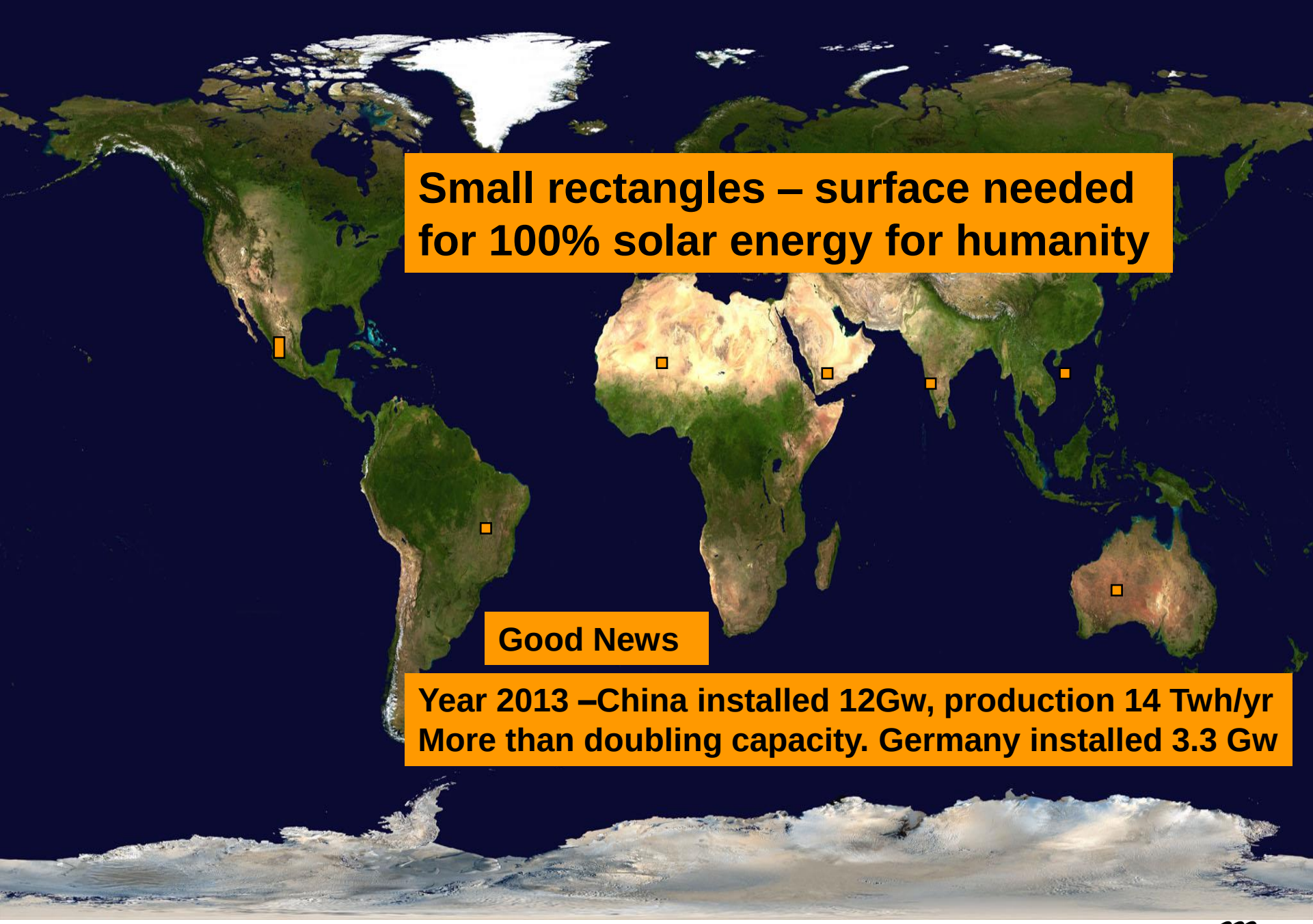
Cheap fast supercharger

DLR ROboMObil

- **experimental electric car**
- **Modelica models**

Tesla model S
range 480 km

Copyright © Open Source Modelica Consortium


Small rectangles – surface needed for 100% solar energy for humanity
Good News
Year 2013 –China installed 12Gw, production 14 Twh/yr
More than doubling capacity. Germany installed 3.3 Gw

Copyright © Open Source Modelica Consortium

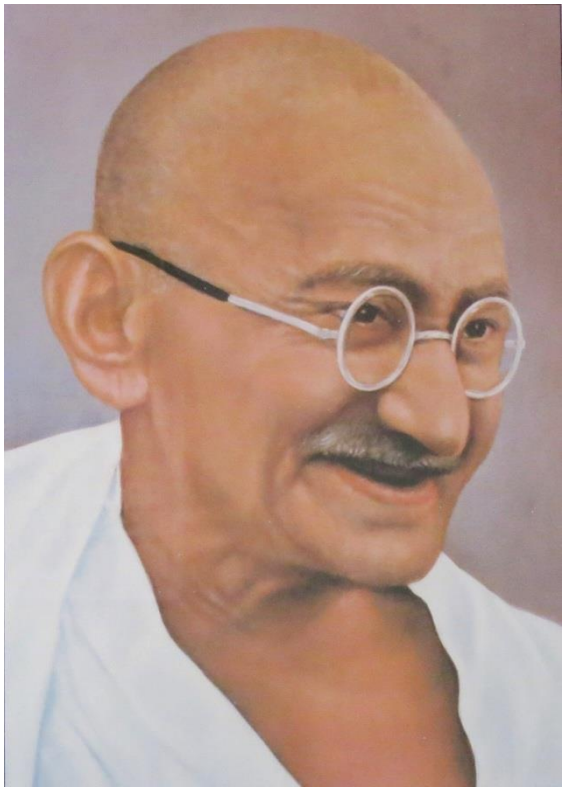
Sustainable Society Necessary for Human Survival

Almost Sustainable

- India, 1.4 ton C02/person/year
- Healthy vegetarian food
- Small-scale agriculture
- Small-scale shops
- Simpler life-style (Mahatma Gandhi)

Non-sustainable

- USA 17 ton CO2, Sweden 7 ton CO2/yr
- High meat consumption (1 kg beef uses ca 4000 L water for production)
- Hamburgers, unhealthy , includes beef
- Energy-consuming mechanized agriculture
- Transport dependent shopping centres
- Stressful materialistic lifestyle



Gandhi – role model for future less materialistic life style

Copyright © Open Source Modelica Consortium

Brief Modelica History

- First Modelica design group meeting in fall 1996
  - International group of people with expert knowledge in both language design and physical modeling
  - Industry and academia
- Modelica Versions
  - 1.0 released September 1997
  - 2.0 released March 2002
  - 2.2 released March 2005
  - 3.0 released September 2007
  - 3.1 released May 2009
  - 3.2 released March 2010
  - 3.3 released May 2012
  - 3.2 rev 2 released November 2013
  - 3.3 rev 1 released July 2014
- Modelica Association established 2000 in Linköping
  - Open, non-profit organization

Copyright © Open Source Modelica Consortium

Modelica Conferences

- The 1st International Modelica conference October, 2000
- The 2nd International Modelica conference March 18-19, 2002
- The 3rd International Modelica conference November 5-6, 2003 in Linköping, Sweden
- The 4th International Modelica conference March 6-7, 2005 in Hamburg, Germany
- The 5th International Modelica conference September 4-5, 2006 in Vienna, Austria
- The 6th International Modelica conference March 3-4, 2008 in Bielefeld, Germany
- The 7th International Modelica conference Sept 21-22, 2009 in Como, Italy
- The 8th International Modelica conference March 20-22, 2011 in Dresden, Germany
- The 9th International Modelica conference Sept 3-5, 2012 in Munich, Germany
- The 10th International Modelica conference March 10-12, 2014 in Lund, Sweden
- Coming: 11th International Modelica conference Sept 21-23, 2015 in Paris

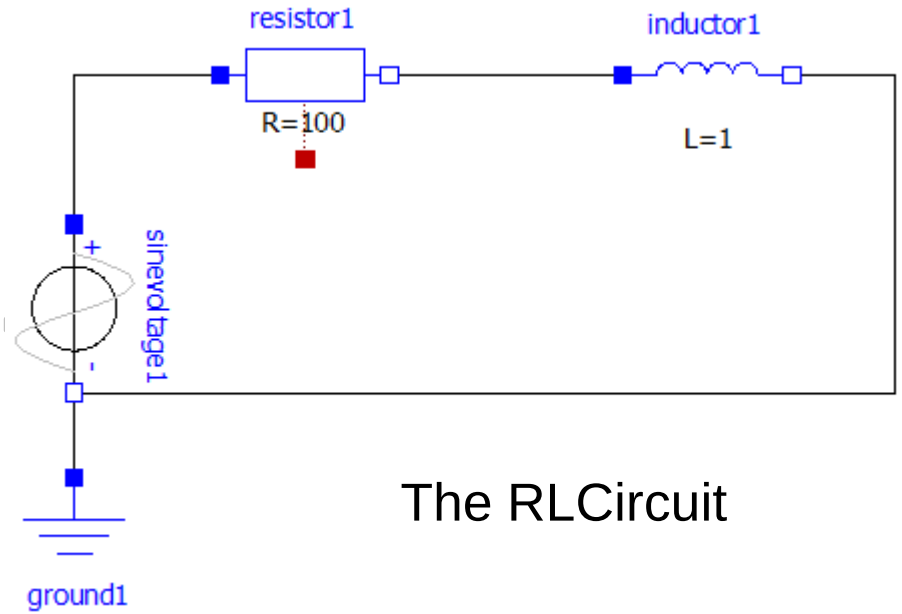
Copyright © Open Source Modelica Consortium

Exercises Part I
Hands-on graphical modeling
(15 minutes)

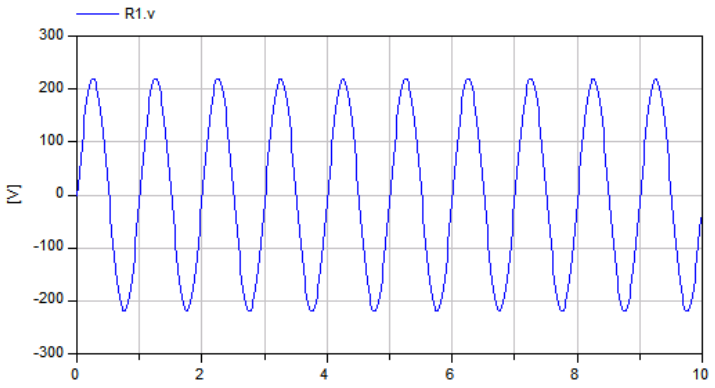
Copyright © Open Source Modelica Consortium

Exercises Part I – Basic Graphical Modeling

- (*See instructions on next two pages*)
- Start the OMEdit editor (part of OpenModelica)
- Draw the RLCircuit
- Simulate



The RLCircuit



Simulation

Copyright © Open Source Modelica Consortium

Exercises Part I – OMEdit Instructions (Part I)

- Start OMEdit from the Program menu under OpenModelica
- Go to **File** menu and choose **New**, and then select **Model**.
- E.g. write *RLCircuit* as the model name.
- For more information on how to use OMEdit, go to **Help** and choose **User Manual** or press **F1**.

- Under the **Modelica Library:**
  - Contains The standard Modelica library components
  - The **Modelica files** contains the list of models you have created.

Copyright © Open Source Modelica Consortium

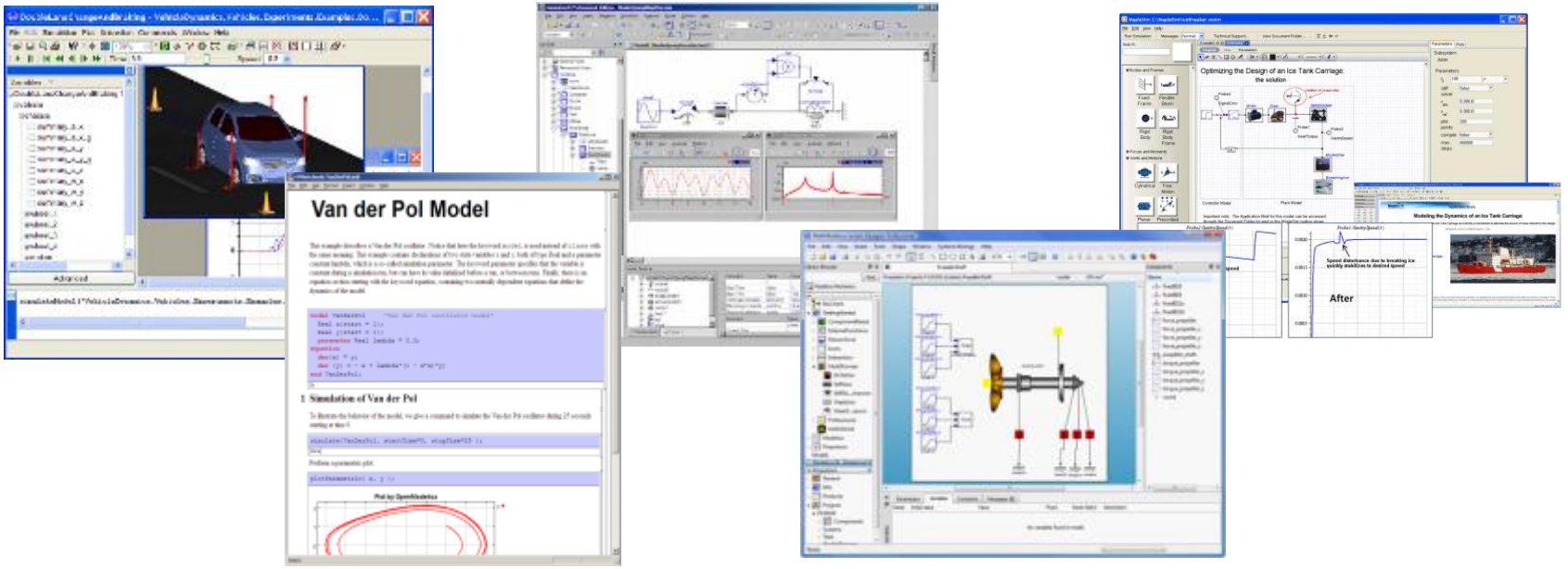
Exercises Part I – OMEdit Instructions (Part II)

- For the RLCircuit model, browse the Modelica standard library and add the following component models:
  - Add `Ground`, `Inductor` and `Resistor` component models from Modelica.Electrical.Analog.Basic package.
  - Add SineVoltage component model from Modelica.Electrical.Analog.Sources package.
- Make the corresponding connections between the component models as shown in slide 38.
- Simulate the model
  - Go to Simulation menu and choose simulate or click on the simulate button in the toolbar.
- Plot the instance variables
  - Once the simulation is completed, a plot variables list will appear on the right side. Select the variable that you want to plot.

Copyright © Open Source Modelica Consortium

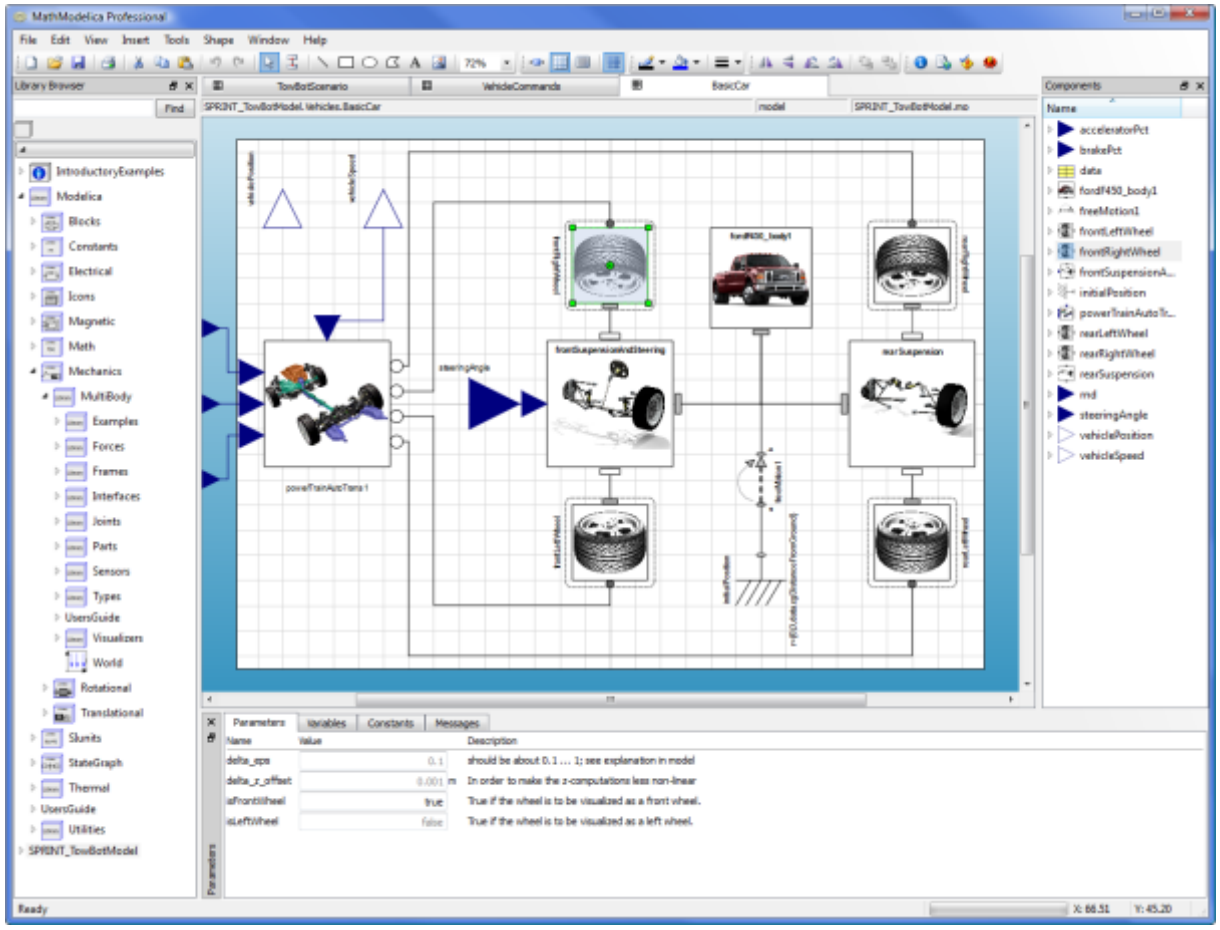
Part II

Modelica environments and OpenModelica



Copyright © Open Source Modelica Consortium

Wolfram System Modeler – Wolfram MathCore

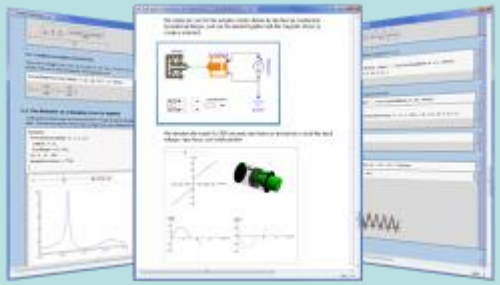


Car model graphical view

Courtesy Wolfram Research

- Wolfram Research
- USA, Sweden
- General purpose
- Mathematica integration
- www.wolfram.com
- www.mathcore.com

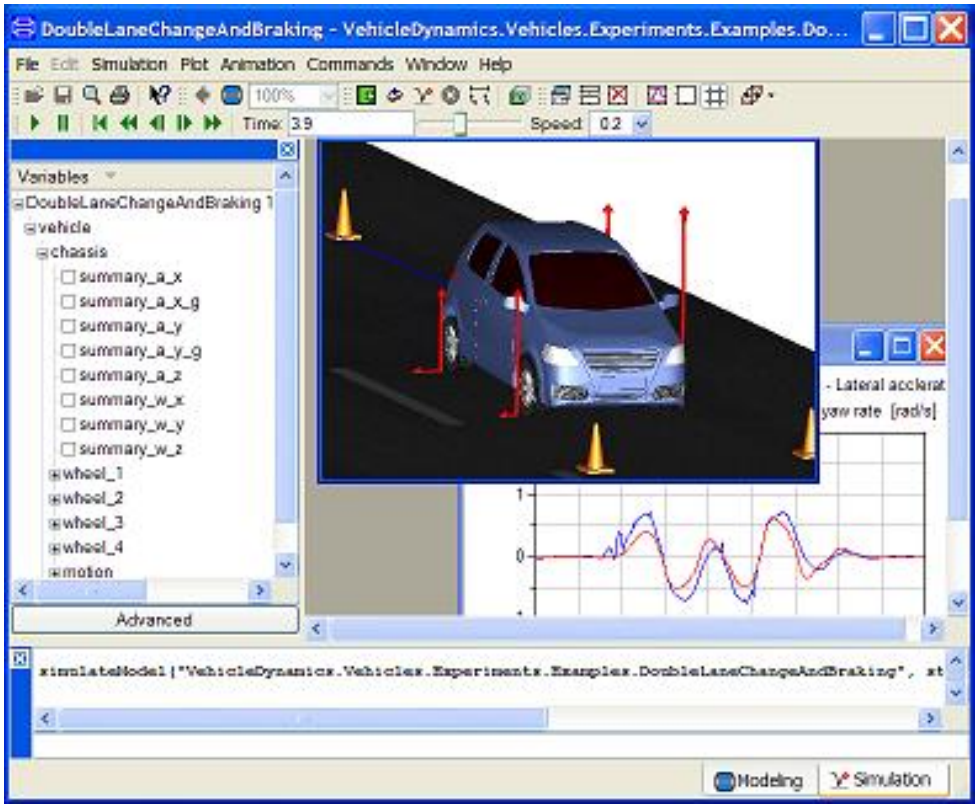
Mathematica



Simulation and analysis

Copyright © Open Source Modelica Consortium

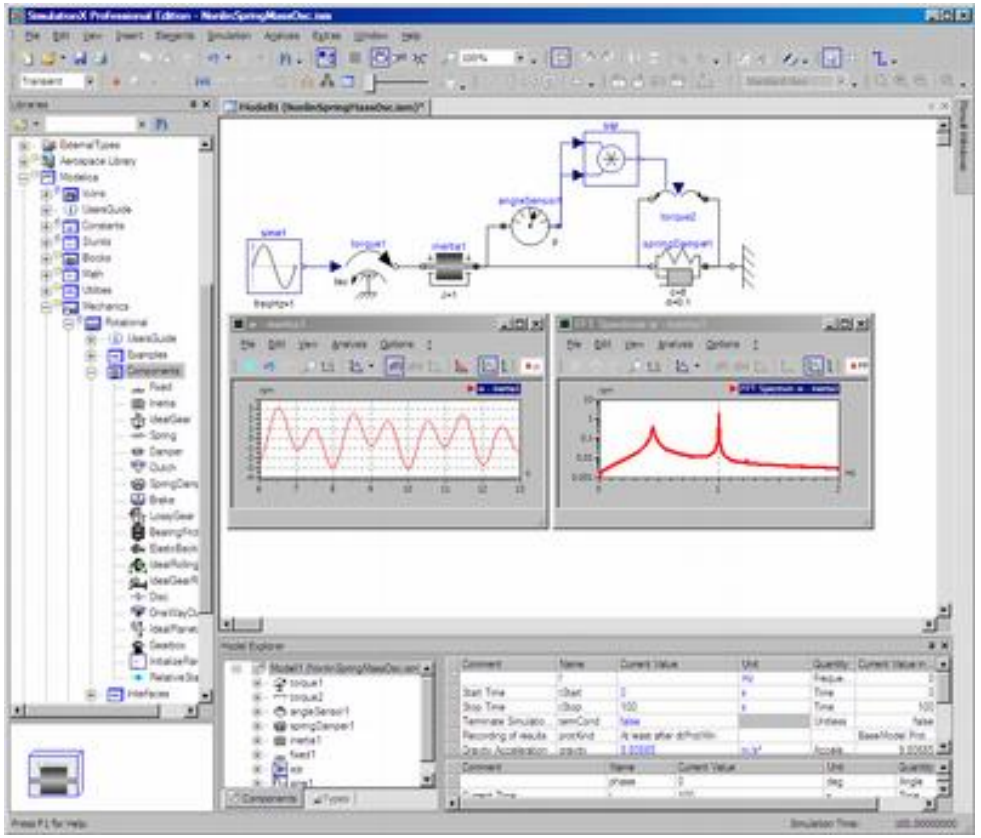
Dymola



- Dassault Systemes Sweden
- Sweden
- First Modelica tool on the market
- Initial main focus on automotive industry
- www.dymola.com

Copyright © Open Source Modelica Consortium

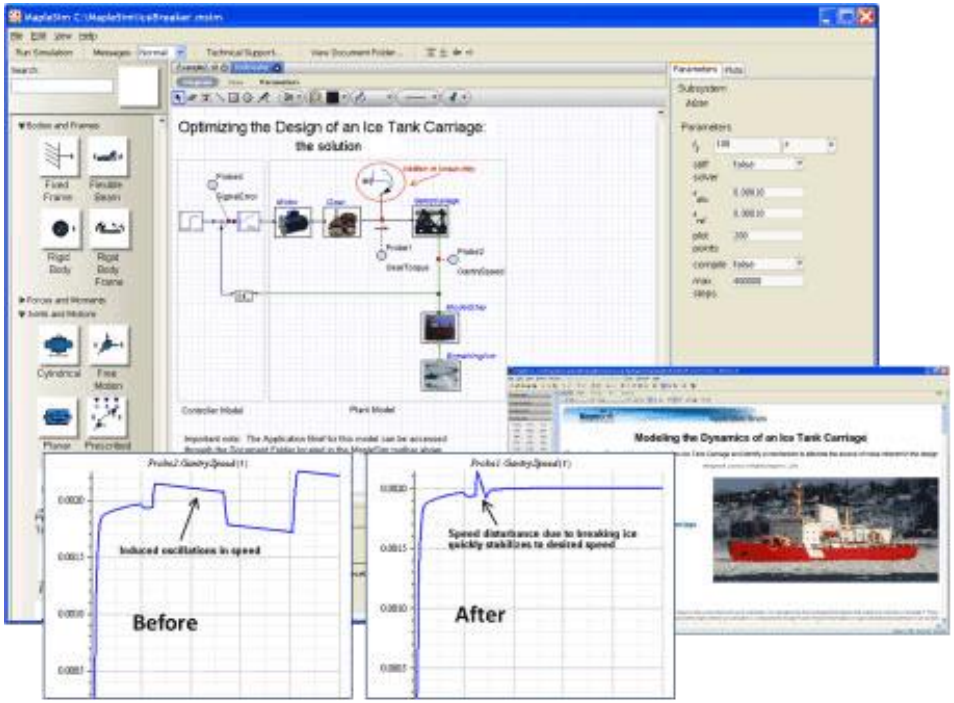
Simulation X



- ITI
- Germany
- Mechatronic systems
- www.simulationx.com

Copyright © Open Source Modelica Consortium

MapleSim

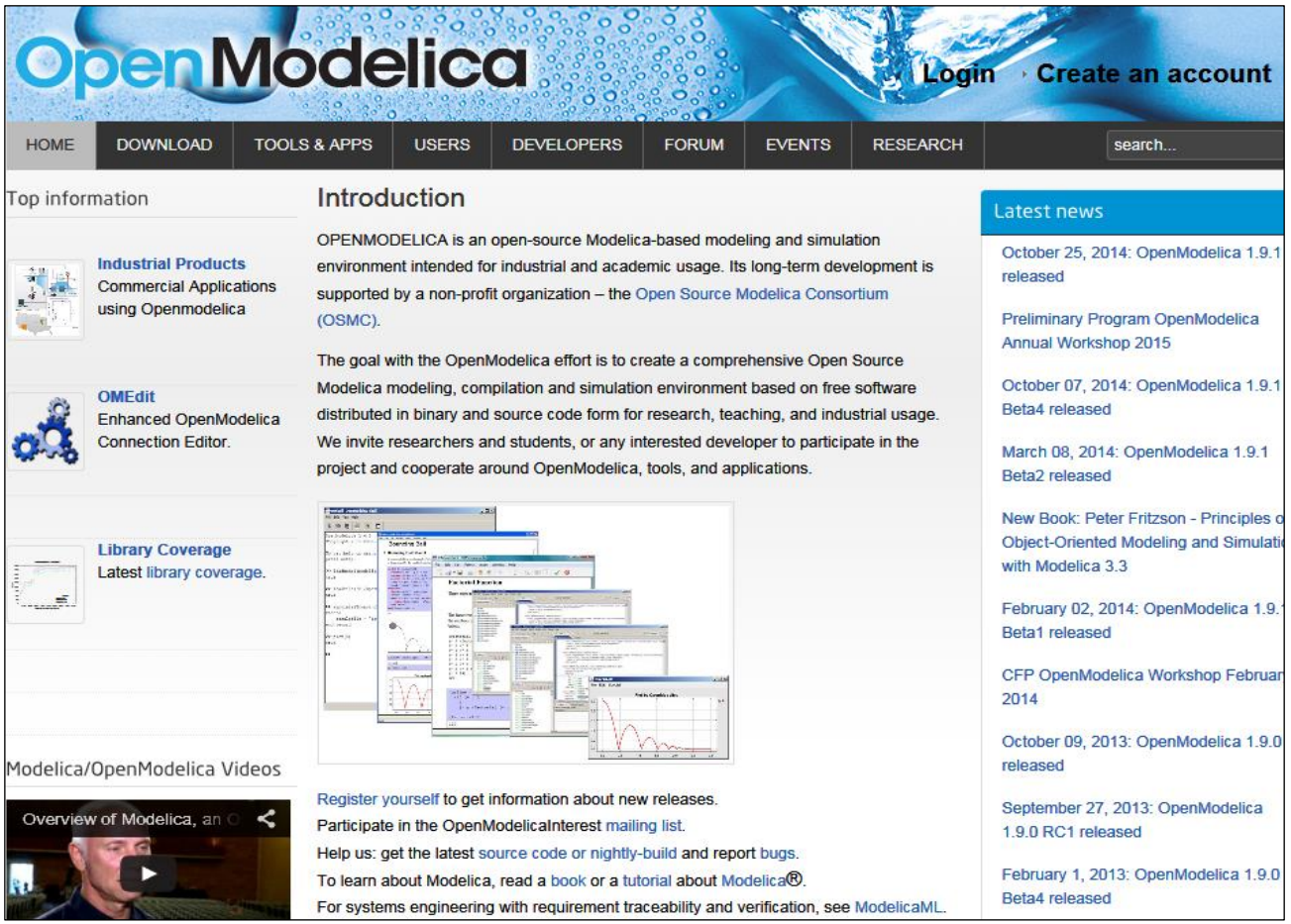

Optimizing the Design of an Ice Tank Carriage:
the solution
Modeling the Dynamics of an Ice Tank Carriage
Induced oscillations in speed
Before
Speed disturbance due to breaking ice quickly stabilizes to desired speed
After

- Maplesoft
- Canada
- Recent Modelica tool on the market
- Integrated with Maple
- www.maplesoft.com

Copyright © Open Source Modelica Consortium

The OpenModelica Environment

www.OpenModelica.org

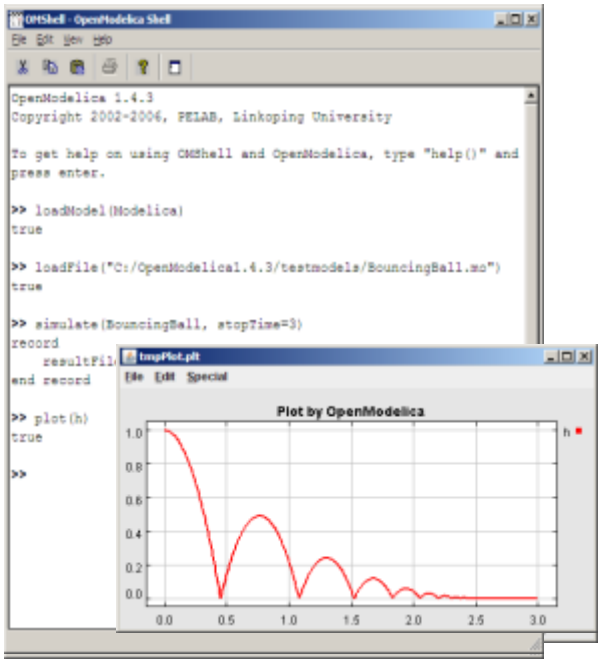

OpenModelica
Login
Create an account
HOME
DOWNLOAD
TOOLS & APPS
USERS
DEVELOPERS
FORUM
EVENTS
RESEARCH
search...
Top information
Industrial Products
Commercial Applications using Openmodelica
OMEdit
Enhanced OpenModelica Connection Editor.
Library Coverage
Latest library coverage.
Modelica/OpenModelica Videos
Overview of Modelica, an
Introduction
OPENMODELICA is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium (OSMC).
The goal with the OpenModelica effort is to create a comprehensive Open Source Modelica modeling, compilation and simulation environment based on free software distributed in binary and source code form for research, teaching, and industrial usage. We invite researchers and students, or any interested developer to participate in the project and cooperate around OpenModelica, tools, and applications.
Register yourself to get information about new releases.
Participate in the OpenModelicaInterest mailing list.
Help us: get the latest source code or nightly-build and report bugs.
To learn about Modelica, read a book or a tutorial about Modelica®.
For systems engineering with requirement traceability and verification, see ModelicaML.
Latest news
October 25, 2014: OpenModelica 1.9.1 released
Preliminary Program OpenModelica Annual Workshop 2015
October 07, 2014: OpenModelica 1.9.1 Beta4 released
March 08, 2014: OpenModelica 1.9.1 Beta2 released
New Book: Peter Fritzson - Principles o Object-Oriented Modeling and Simulati with Modelica 3.3
February 02, 2014: OpenModelica 1.9.1 Beta1 released
CFP OpenModelica Workshop Februar 2014
October 09, 2013: OpenModelica 1.9.0 released
September 27, 2013: OpenModelica 1.9.0 RC1 released
February 1, 2013: OpenModelica 1.9.0 Beta4 released

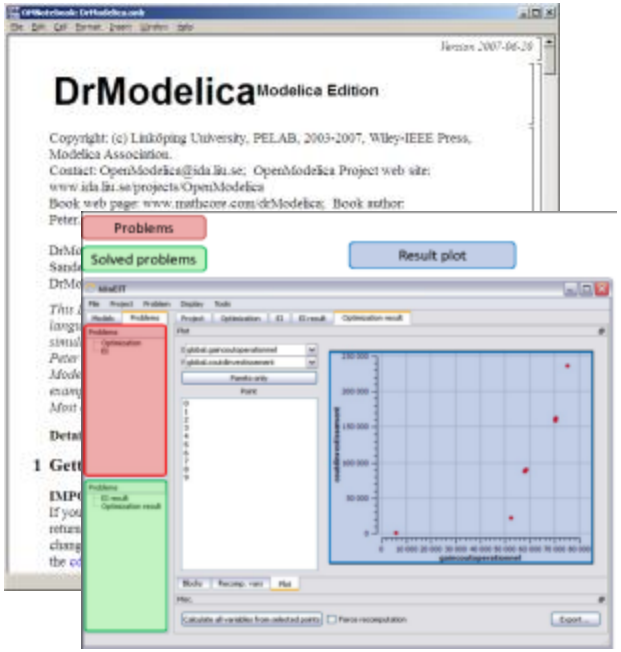
Copyright © Open Source Modelica Consortium

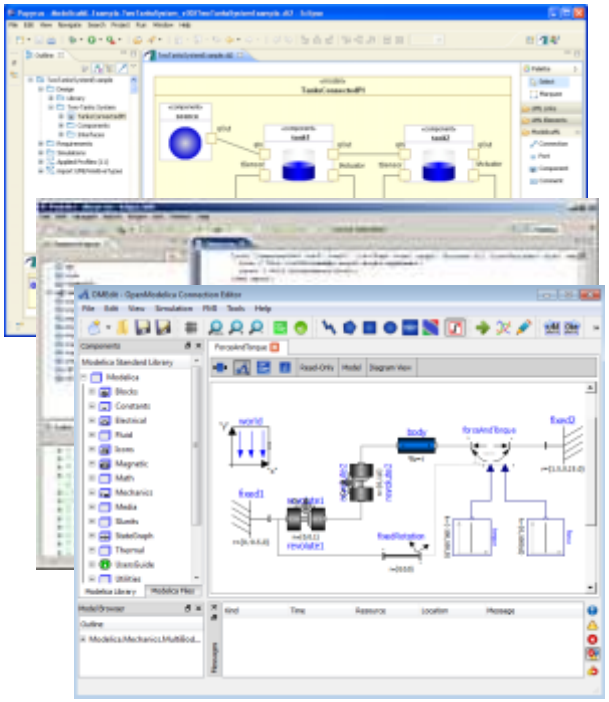
The OpenModelica Open Source Environment
www.openmodelica.org

- Advanced Interactive Modelica compiler (OMC)
  - Supports most of the Modelica Language
  - **Modelica** and **Python scripting**
- Basic environment for creating models
  - **OMShell** – an interactive command handler
  - **OMNotebook** – a literate programming notebook
  - **MDT** – an advanced textual environment in Eclipse

- **OMEdit** graphic Editor
- **OMDebugger** for equations
- **OMOptim** optimization tool
- **OM Dynamic optimizer** collocation
- **ModelicaML** UML Profile
- **MetaModelica** extension
- **ParModelica** extension







Copyright © Open Source Modelica Consortium

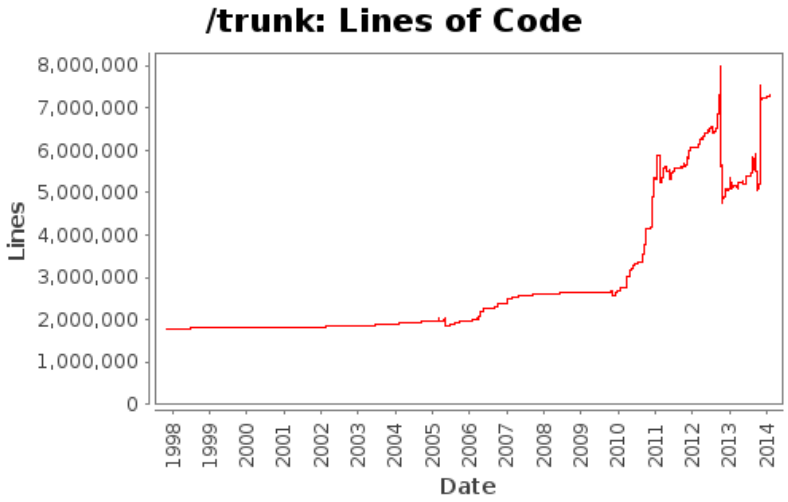
OSMC – International Consortium for Open Source Model-based Development Tools, 46 members Mar 2014

Founded Dec 4, 2007

Open-source community services

- **Website and Support Forum**
- **Version-controlled source base**
- **Bug database**
- **Development courses**
- **www.openmodelica.org**

Code Statistics



Industrial members

- **Bosch Rexroth AG, Germany**
- **Siemens PLM, California, USA**
- **Siemens Turbo, Sweden**
- **CDAC Centre, Kerala, India**
- **Creative Connections, Prague**
- **DHI, Aarhus, Denmark**
- **EDF, Paris, France**
- **Equa Simulation AB, Sweden**
- **Fraunhofer IWES, Bremerhaven**
- **Frontway AB, Sweden**
- **IFP, Paris, France**
- **GTI, USA**
- **ISID Dentsu, Tokyo, Japan**
- **ITI, Dresden, Germany**
- **Maplesoft, Canada**
- **Ricardo Inc., USA**
- **STEAG, Dehli, India**
- **TLK Thermo, Germany**
- **Sozhou Tongyuan, China**
- **VTI, Linköping, Sweden**
- **VTT, Finland**
- **Wolfram MathCore, Sweden**

University members

- **Austrian Inst. of Tech, Austria**
- **Linköping University, Sweden**
- **UC Berkeley, USA**
- **TU Berlin, Insti UEBB, Germany**
- **FH Bielefeld, Bielefeld, Germany**
- **TU Braunschweig, Germany**
- **Univ Calabria, Italy**
- **Danish Technical Univ, Denmark**
- **TU Dortmund, Germany**
- **TU Dresden, Germany**
- **Université Laval, Canada**
- **Georgia Inst. Technology, USA**
- **Ghent University, Belgium**
- **Halmstad University, Sweden**
- **Heidelberg University, Germany**
- **TU Hamburg/Harburg Germany**
- **KTH, Stockholm, Sweden**
- **Univ of Maryland, Syst Eng USA**
- **Univ of Maryland, CEEE, USA**
- **Politecnico di Milano, Italy**
- **Ecoles des Mines, CEP, France**
- **Mälardalen University, Sweden**
- **Univ Pisa, Italy**
- **Telemark Univ College, Norway**

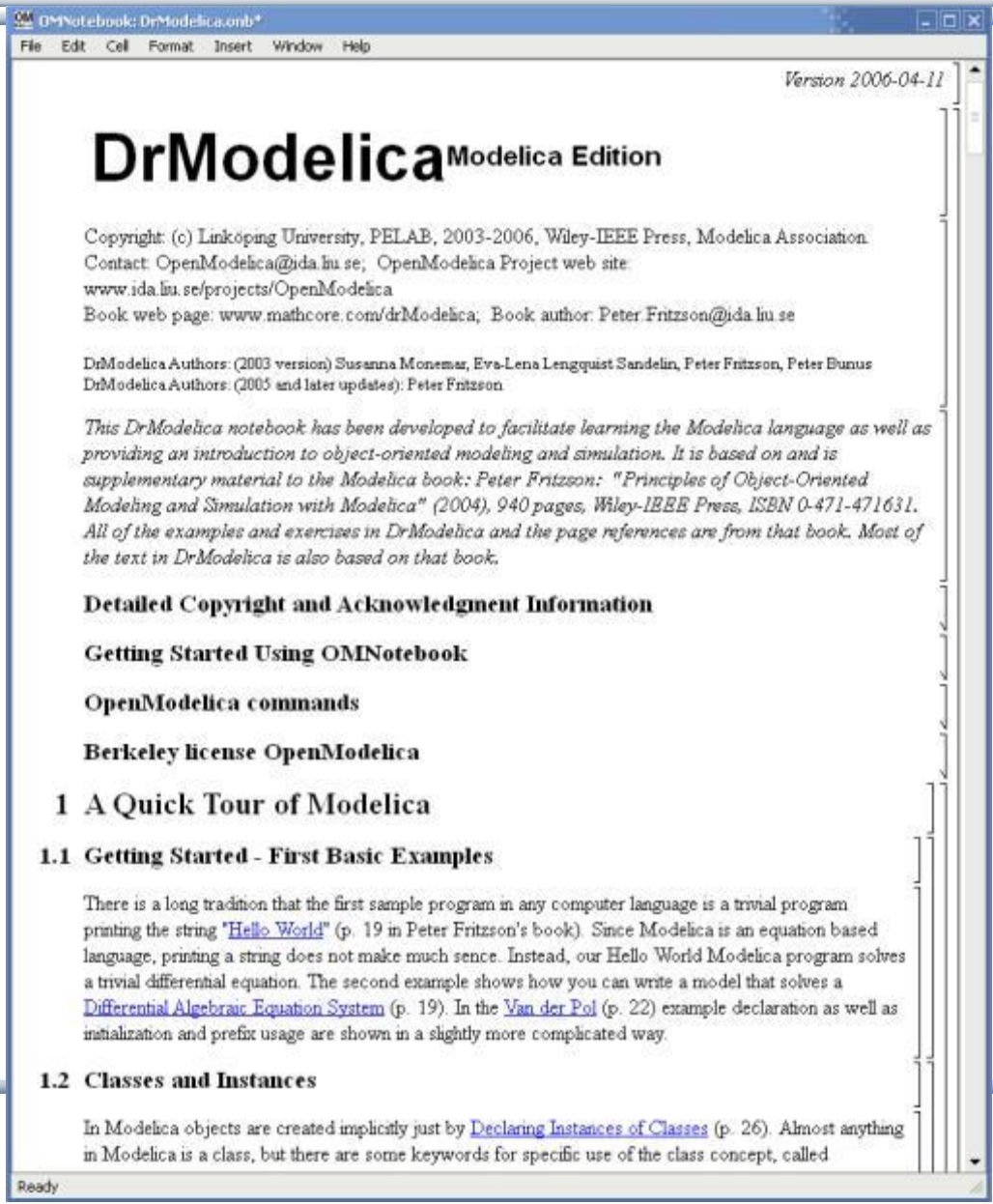
Copyright © Open Source Modelica Consortium

OMNotebook Electronic Notebook with DrModelica

- Primarily for teaching
- Interactive electronic book
- Platform independent

Commands:

- *Shift-return (evaluates a cell)*
- File Menu (open, close, etc.)
- Text Cursor (vertical), Cell cursor (horizontal)
- Cell types: text cells & executable code cells
- Copy, paste, group cells
- Copy, paste, group text
- Command Completion (shift-tab)


OMNotebook: DrModelica.onb*
File Edit Cell Format Insert Window Help
Version 2006-04-11
DrModelica Modelica Edition
Copyright: (c) Linköping University, PELAB, 2003-2006, Wiley-IEEE Press, Modelica Association
Contact: OpenModelica@ida.liu.se; OpenModelica Project web site:
www.ida.liu.se/projects/OpenModelica
Book web page: www.mathcore.com/drModelica; Book author: Peter.Fritzson@ida.liu.se
DrModelica Authors: (2003 version) Susanna Monemar, Eva-Lena Lengquist Sandelin, Peter Fritzson, Peter Bunus
DrModelica Authors: (2005 and later updates): Peter Fritzson
This DrModelica notebook has been developed to facilitate learning the Modelica language as well as providing an introduction to object-oriented modeling and simulation. It is based on and is supplementary material to the Modelica book: Peter Fritzson: "Principles of Object-Oriented Modeling and Simulation with Modelica" (2004), 940 pages, Wiley-IEEE Press, ISBN 0-471-471631. All of the examples and exercises in DrModelica and the page references are from that book. Most of the text in DrModelica is also based on that book.
Detailed Copyright and Acknowledgment Information
Getting Started Using OMNotebook
OpenModelica commands
Berkeley license OpenModelica
1 A Quick Tour of Modelica
1.1 Getting Started - First Basic Examples
There is a long tradition that the first sample program in any computer language is a trivial program printing the string "Hello World" (p. 19 in Peter Fritzson's book). Since Modelica is an equation based language, printing a string does not make much sence. Instead, our Hello World Modelica program solves a trivial differential equation. The second example shows how you can write a model that solves a Differential Algebraic Equation System (p. 19). In the Van der Pol (p. 22) example declaration as well as initialization and prefix usage are shown in a slightly more complicated way.
1.2 Classes and Instances
In Modelica objects are created implicitly just by Declaring Instances of Classes (p. 26). Almost anything in Modelica is a class, but there are some keywords for specific use of the class concept, called
Ready

Copyright © Open Source Modelica Consortium

OMnotebook Interactive Electronic Notebook Here Used for Teaching Control Theory

OMNotebook: Kalman.onb

File Edit Cell Format Insert Window Help

1 Kalman Filter

Often we don't have access to the internal states of
have to reconstruct the state of the system based o
The idea with an observer is that we feedback the
the estiamtion is correct then the difference should

Another difficulty is that the measured quantities of

$$\begin{cases} \dot{\hat{x}} = \\ \end{cases}$$

Here are e denoting a disturbance in the input signa
be evaluated by the difference

$$K(y(t)$$

By using this quantity as feedback we obtain the ob

$$\dot{\hat{x}} = A\hat{x}(t) + Bu(t$$

Now form the error as

The differential error is

Ready

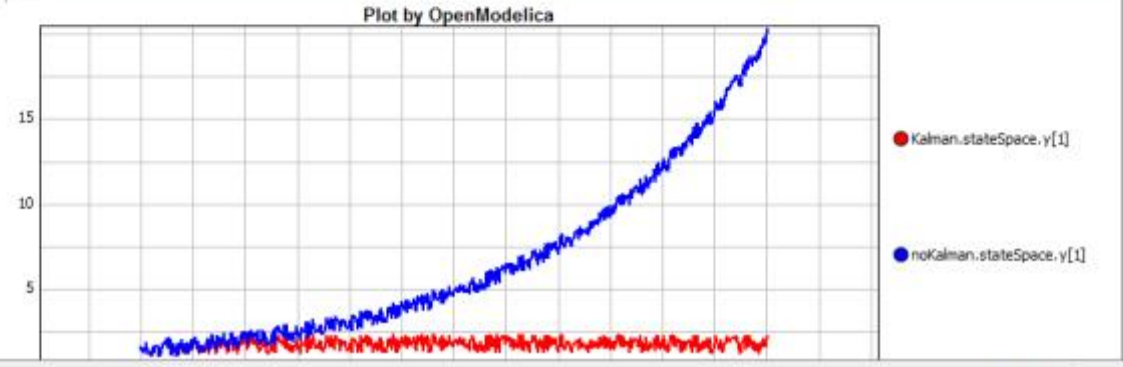
OMNotebook: Kalman.onb*

File Edit Cell Format Insert Window Help

```
model KalmanFeedback
  parameter Real A[:,size(A, 1)] = {{0,1},{1,0}} ;
  parameter Real B[size(A, 1),:] = {{0},{1}};
  parameter Real C[:,size(A, 1)] = {{1,0}};
  parameter Real[2,1] K = [2.4;3.4];
  parameter Real[1,2] L = [2.4,3.4];
  parameter Real[:,:] ABL = A-B*L;
  parameter Real[:,:] BL = B*L;
  parameter Real[:,:] Z = zeros(size(ABL,2),size(AKC,1));
  parameter Real[:,:] AKC = A-K*C;
  parameter Real[:,:] Anew = [0,1,0,0 ; -1.4, -3.4, 2.4,3.4; 0,0,-2.4,1;0,0,-2.4,0];
  parameter Real[:,:] Bnew = [0;1;0;0];
  parameter Real[:,:] Fnew = [1;0;0;0];
  stateSpaceNoise Kalman(stateSpace.A=Anew,stateSpace.B=Bnew,  stateSpace.C=[1,0,0,0],
stateSpace.F = Fnew);
  stateSpaceNoise noKalman;
end KalmanFeedback;
```

```
simulate(KalmanFeedback,stopTime=3)
plot({Kalman.stateSpace.y[1],noKalman.stateSpace.y[1]})
```

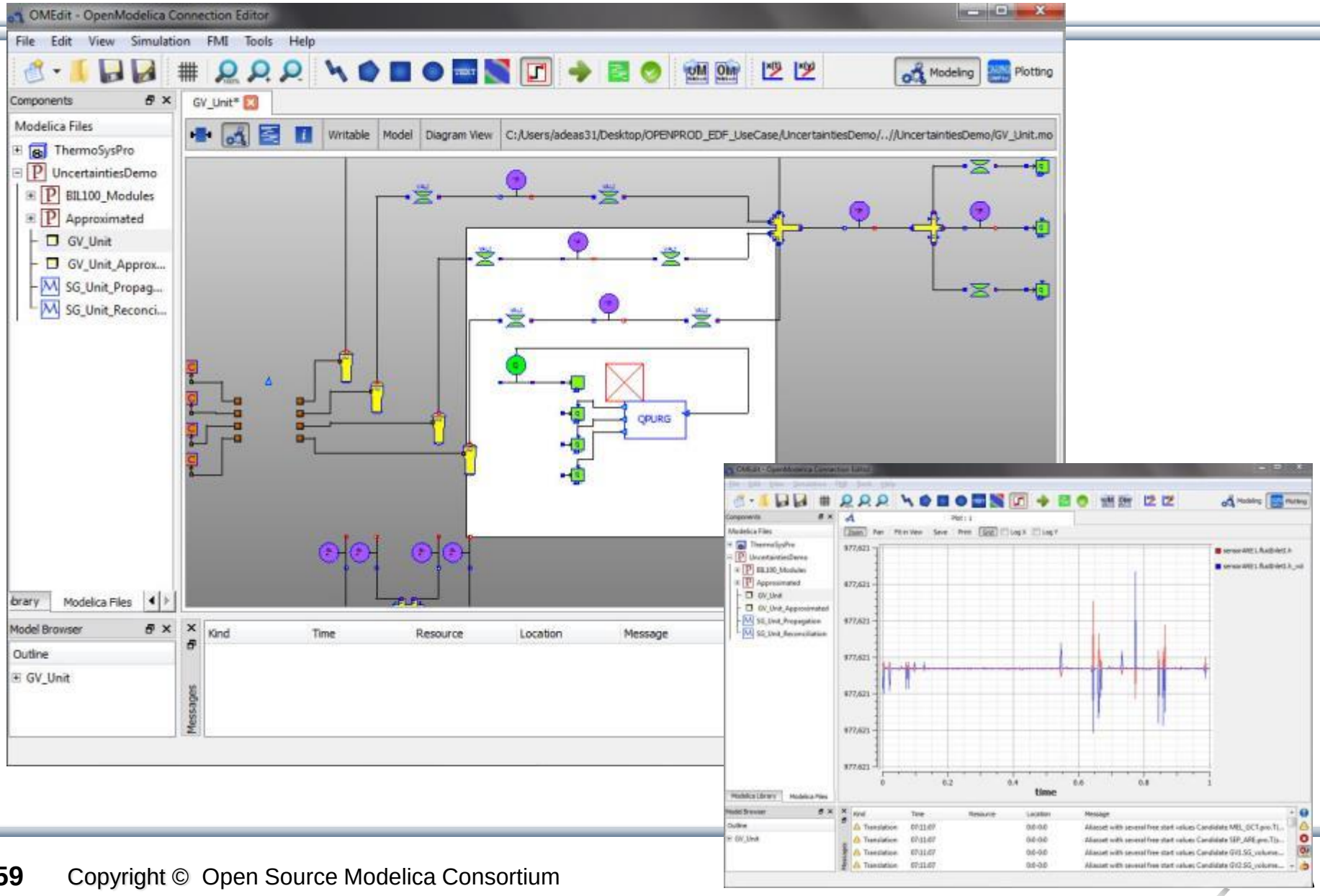
true



Ready Ln 12, Col 39

Copyright © Open Source Modelica Consortium

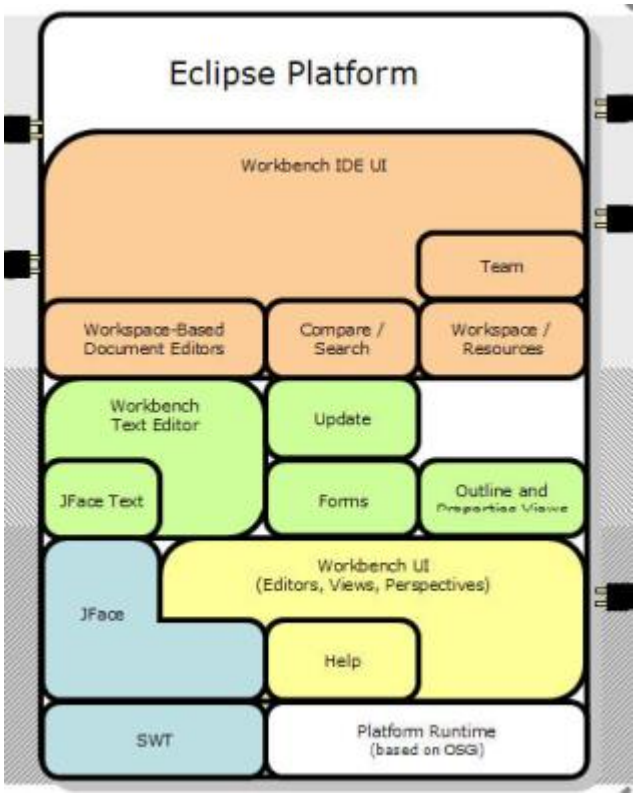
OpenModelica Environment Demo



Copyright © Open Source Modelica Consortium

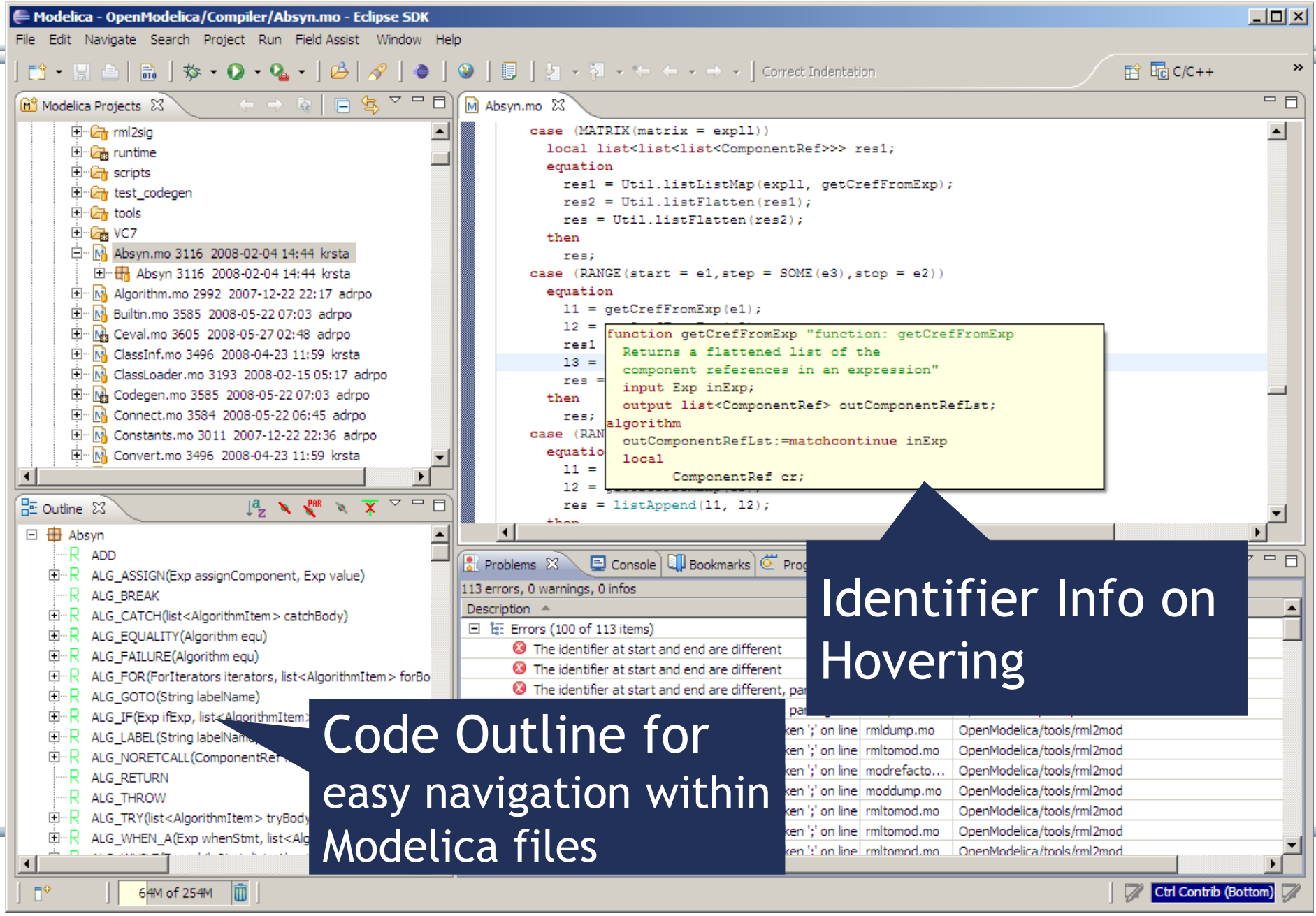
OpenModelica MDT – Eclipse Plugin

- Browsing of packages, classes, functions
- Automatic building of executables; separate compilation
- Syntax highlighting
- Code completion,
  Code query support for developers
- Automatic Indentation
- Debugger
  (Prel. version for algorithmic subset)

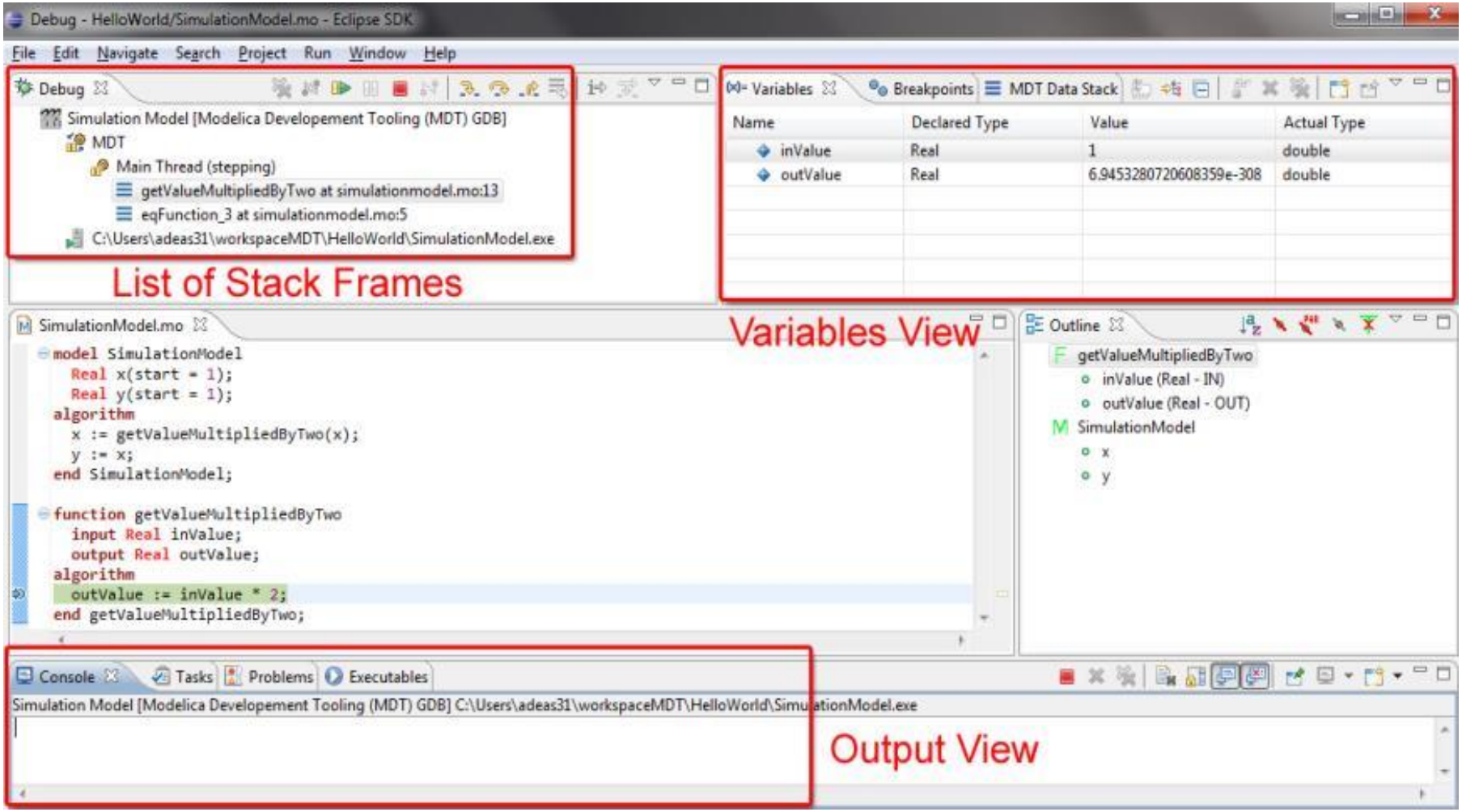


Copyright © Open Source Modelica Consortium

OpenModelica MDT: Code Outline and Hovering Info



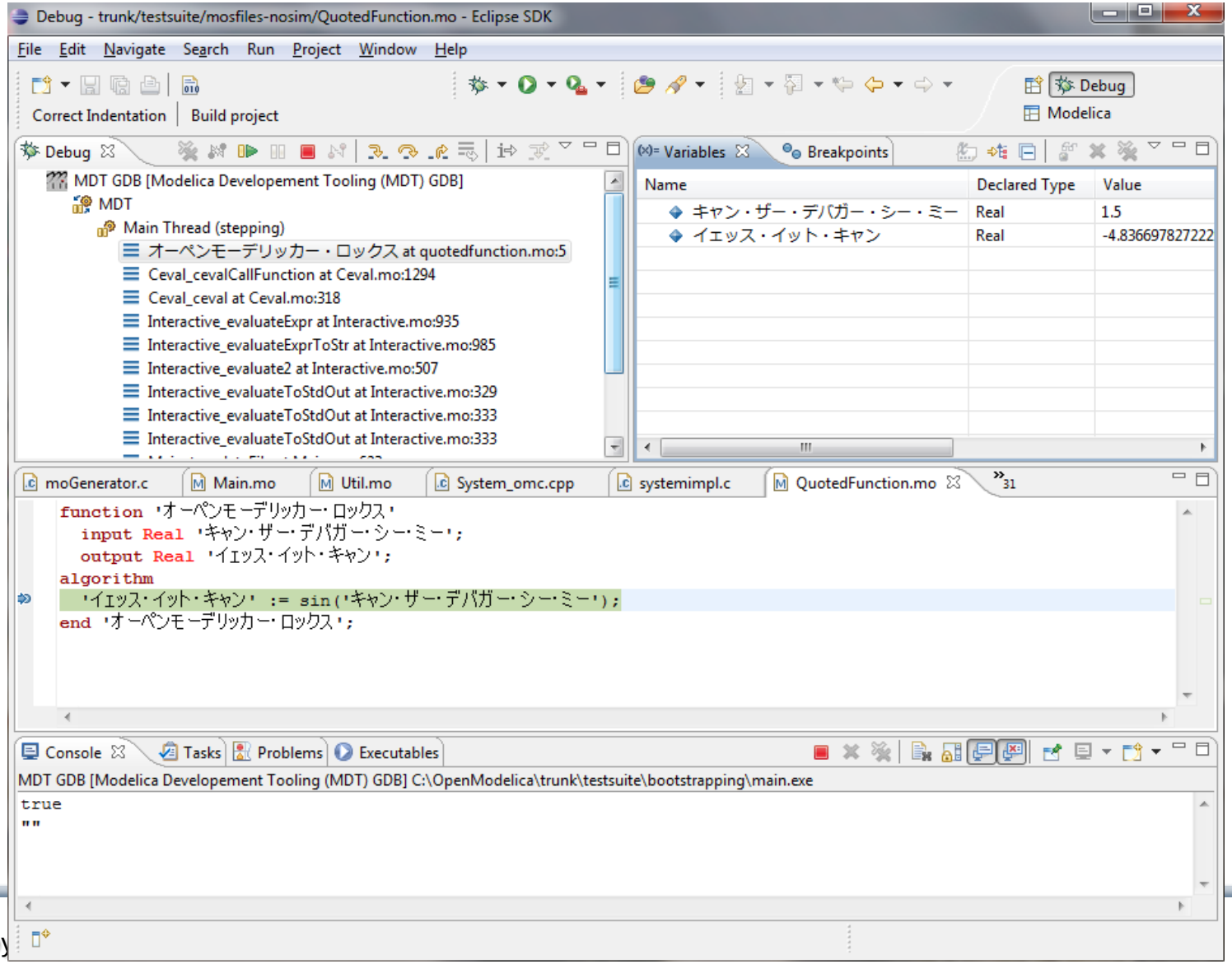
OpenModelica MDT Algorithmic Code Debugger



Copyright © Open Source Modelica Consortium

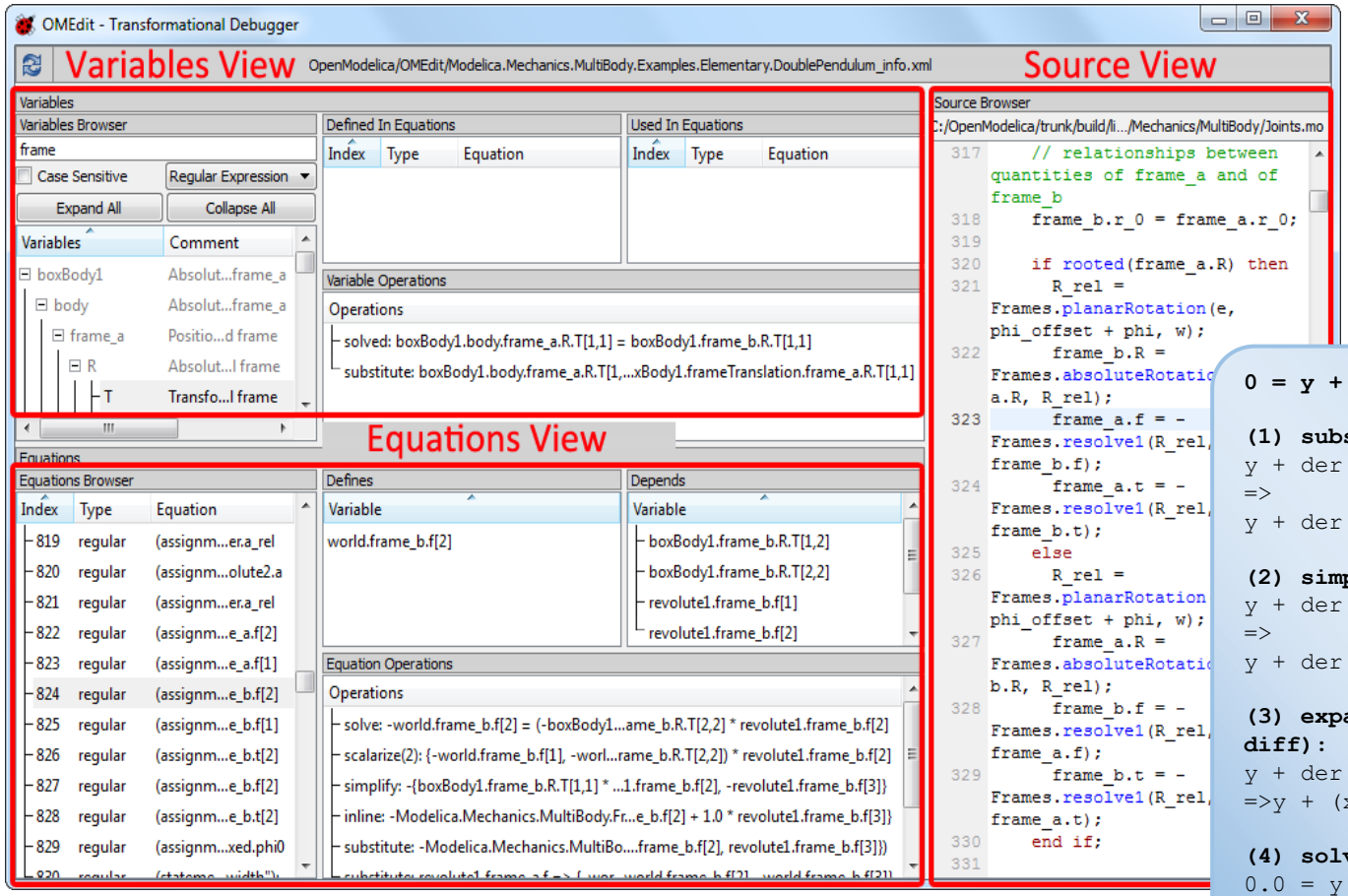
The OpenModelica MDT Debugger (Eclipse-based)

Using Japanese Characters



Copy

OpenModelica Equation Model Debugger



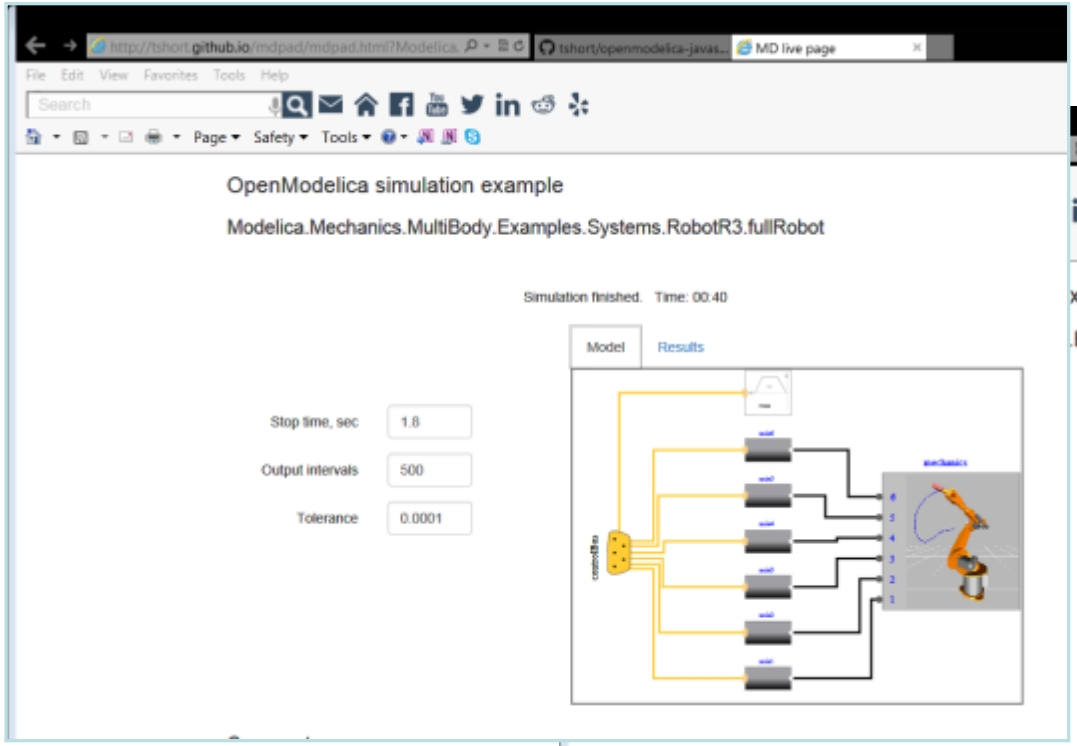
Showing equation transformations of a model:

```
0 = y + der(x * time * z); z = 1.0;

(1) substitution:
y + der(x * (time * z))
=>
y + der(x * (time * 1.0))

(2) simplify:
y + der(x * (time * 1.0))
=>
y + der(x * time)

(3) expand derivative (symbolic diff):
y + der(x * time)
=>y + (x + der(x) * time)

(4) solve:
0.0 = y + (x + der(x) * time)
=>
der(x) = ((-y) - x) / time
time <> 0
```

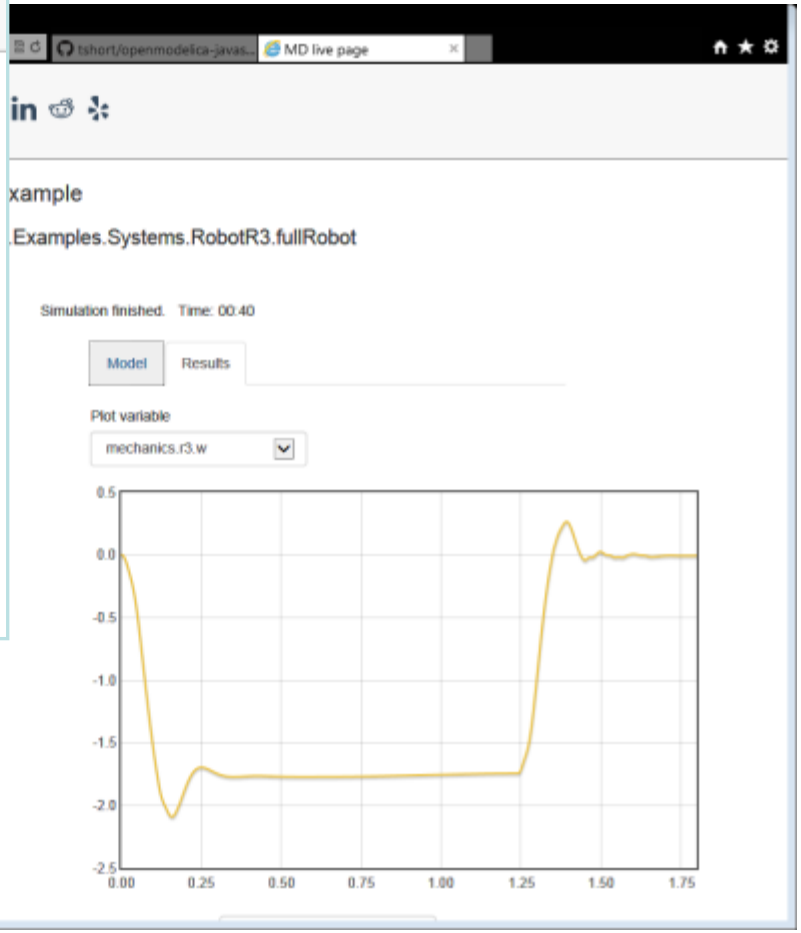
Mapping run-time error to source model position

Copyright © Open Source Modelica Consortium

OpenModelica Simulation in Web Browser Client



MultiBody RobotR3.FullRobot

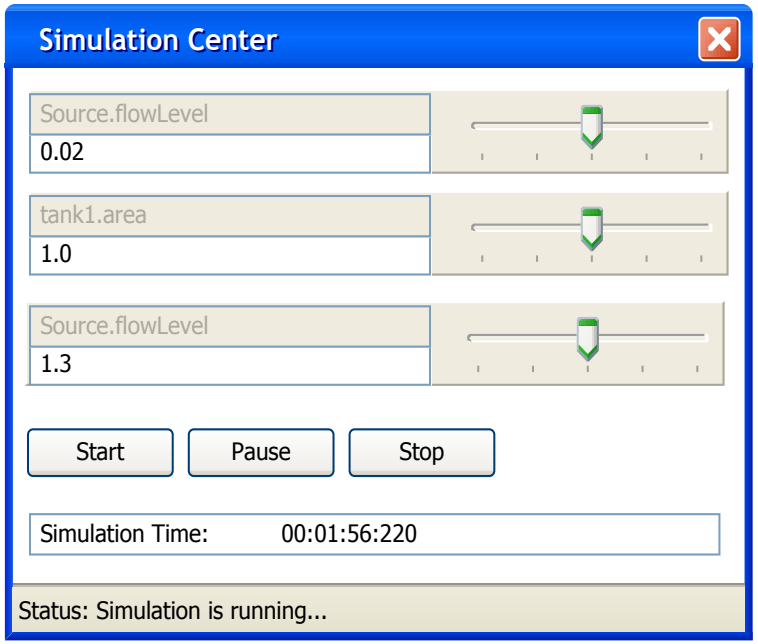


OpenModelica compiles to efficient Java Script code which is executed in web browser

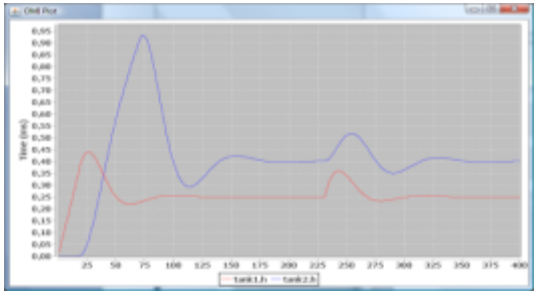
Copyright © Open Source Modelica Consortium

Interactive Simulation

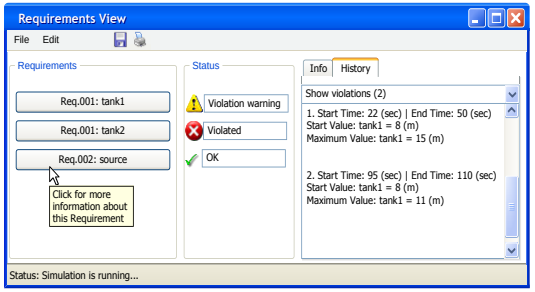
Simulation Control



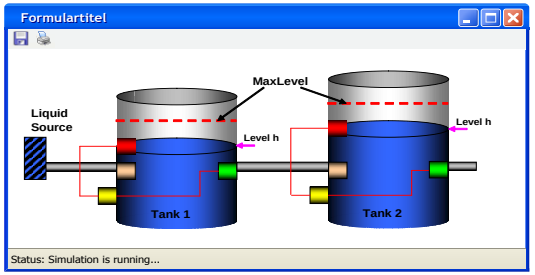
Examples of Simulation Visualization



Plot View



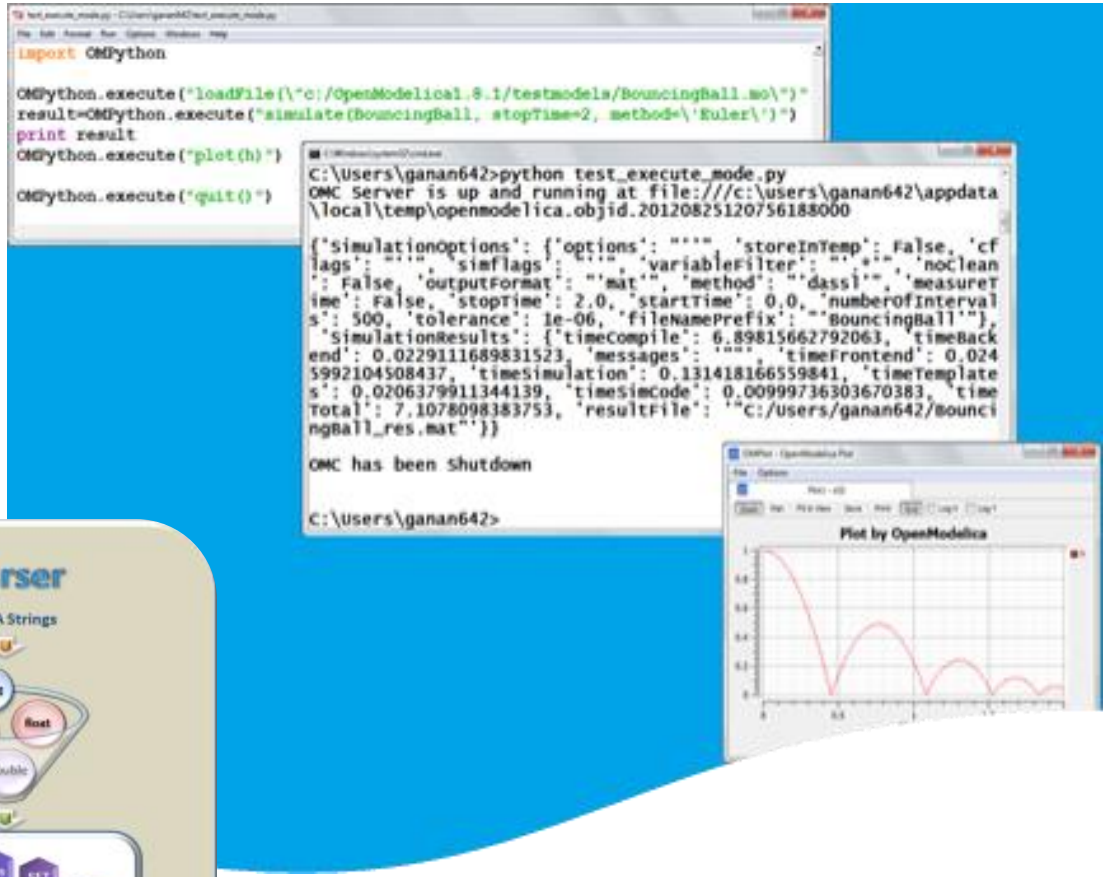
Requirements Evaluation View in ModelicaML

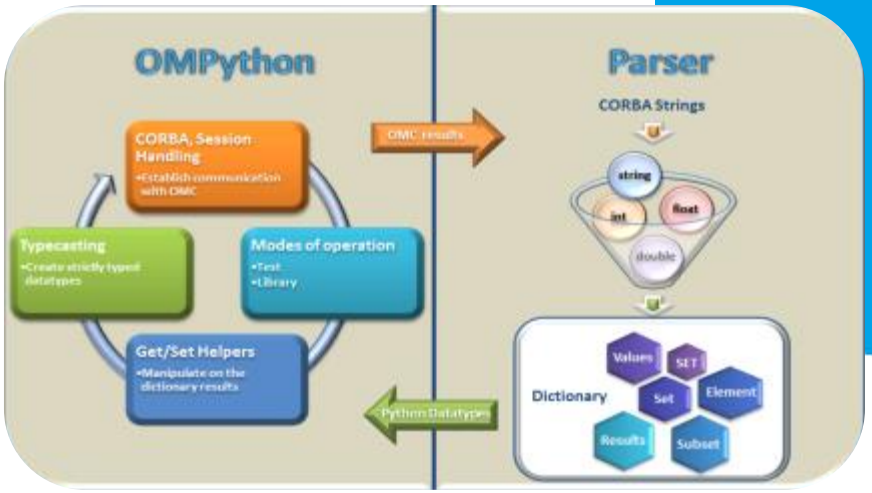


Domain-Specific Visualization View

OMPython – Python Scripting with OpenModelica

- Interpretation of Modelica commands and expressions
- Interactive Session handling
- Library / Tool
- Optimized Parser results
- Helper functions
- Deployable, Extensible and Distributable

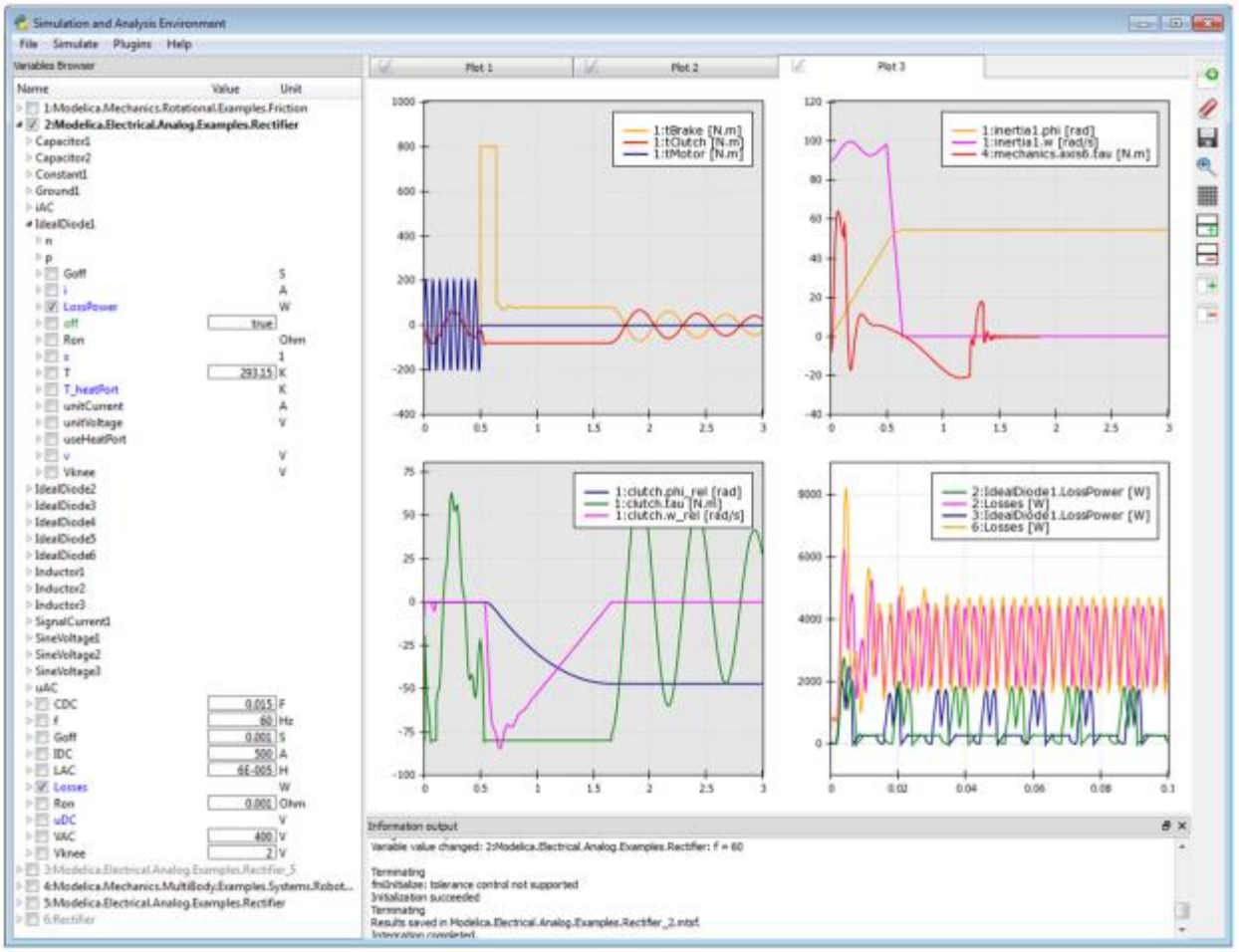




Copyright © Open Source Modelica Consortium

PySimulator Package

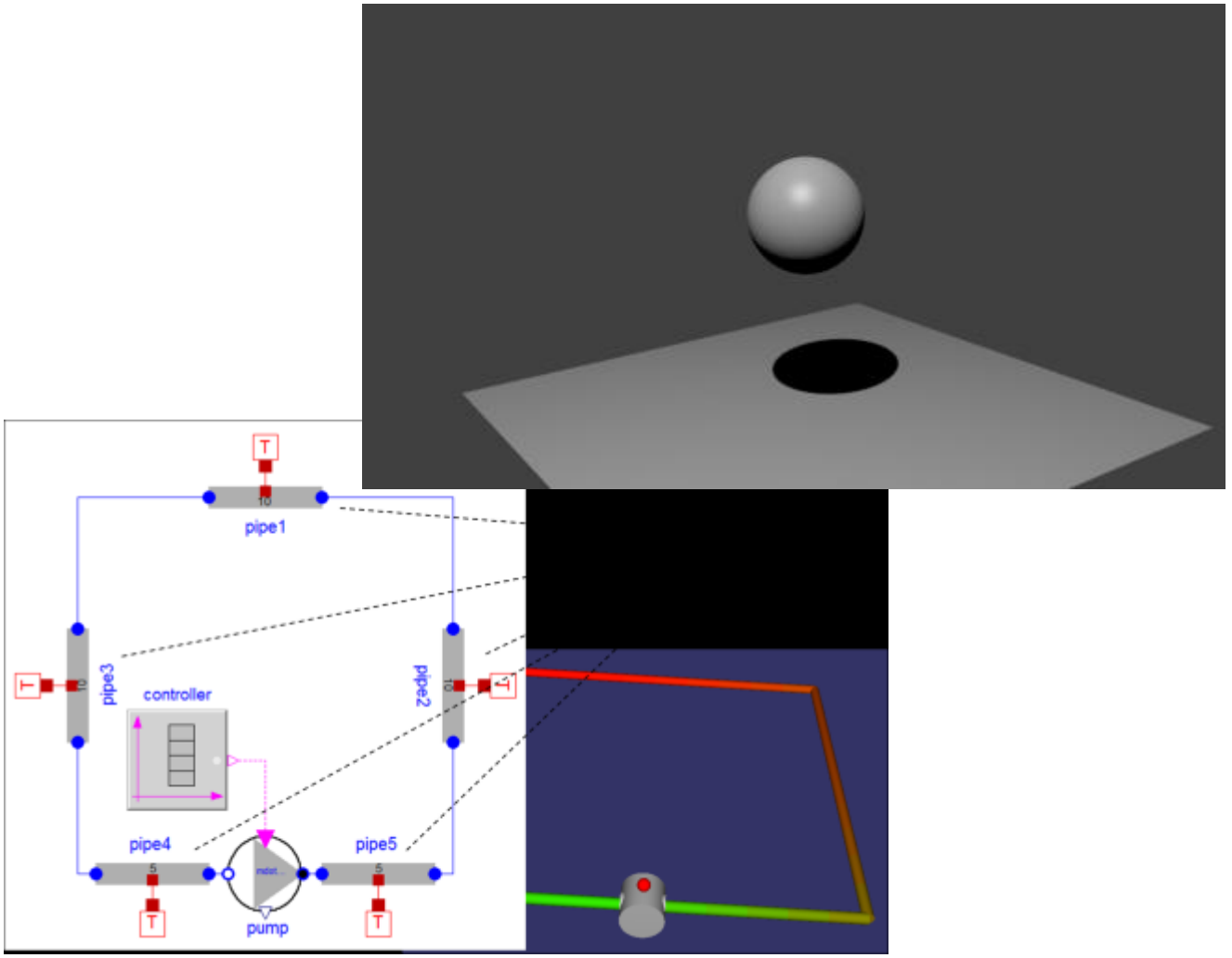
- PySimulator, a simulation and analysis package developed by DLR
- Free, downloadable
- Uses OMPython to simulate Modelica models by OpenModelica



Copyright © Open Source Modelica Consortium

Modelica3D Library

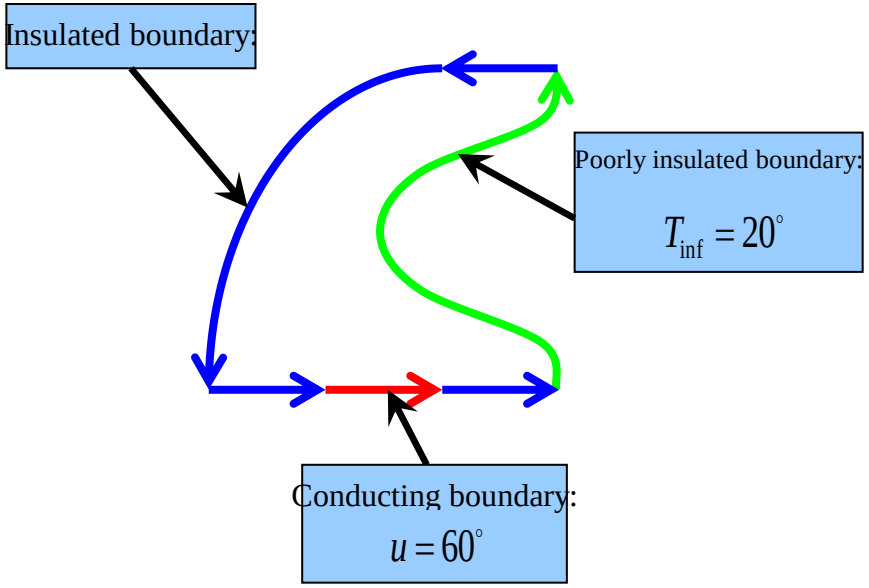
- Modelica 3D Graphics Library by Fraunhofer FIRST, Berlin
- Part of OpenModelica distribution
- Can be used for 3D graphics in OpenModelica

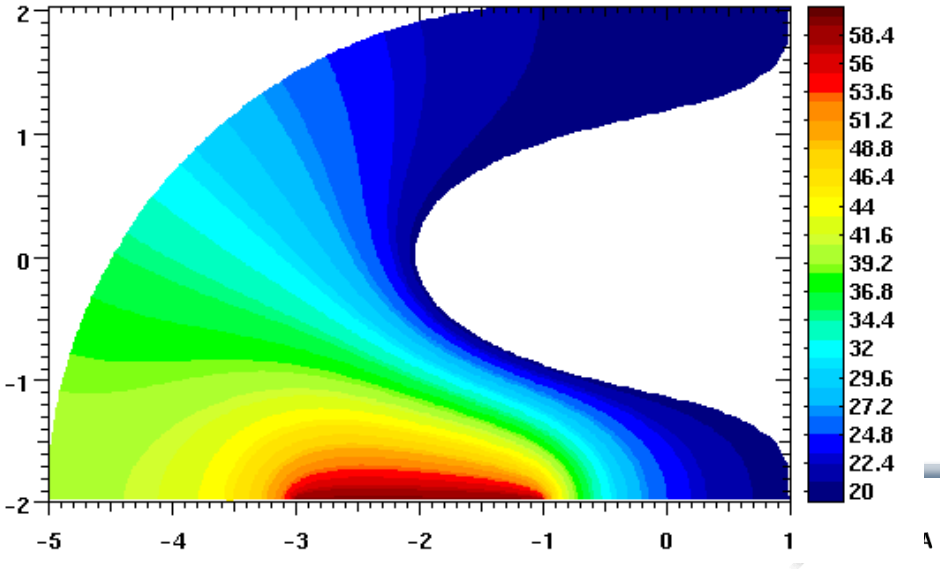


Copyright © Open Source Modelica Consortium

Extending Modelica with PDEs for 2D, 3D flow problems – Research

```
class PDEModel
   HeatNeumann h_iso;
   Dirichlet h_heated(g=50);
   HeatRobin h_glass(h_heat=30000);
   HeatTransfer ht;
   Rectangle2D dom;
equation
   dom.eq=ht;
   dom.left.bc=h_glass;
   dom.top.bc=h_iso;
   dom.right.bc=h_iso;
   dom.bottom.bc=h_heated;
end PDEModel;
```





Prototype in OpenModelica 2005
PhD Thesis by Levon Saldamli
www.openmodelica.org
Currently not operational

Copyright © Open Source Modelica Consortium

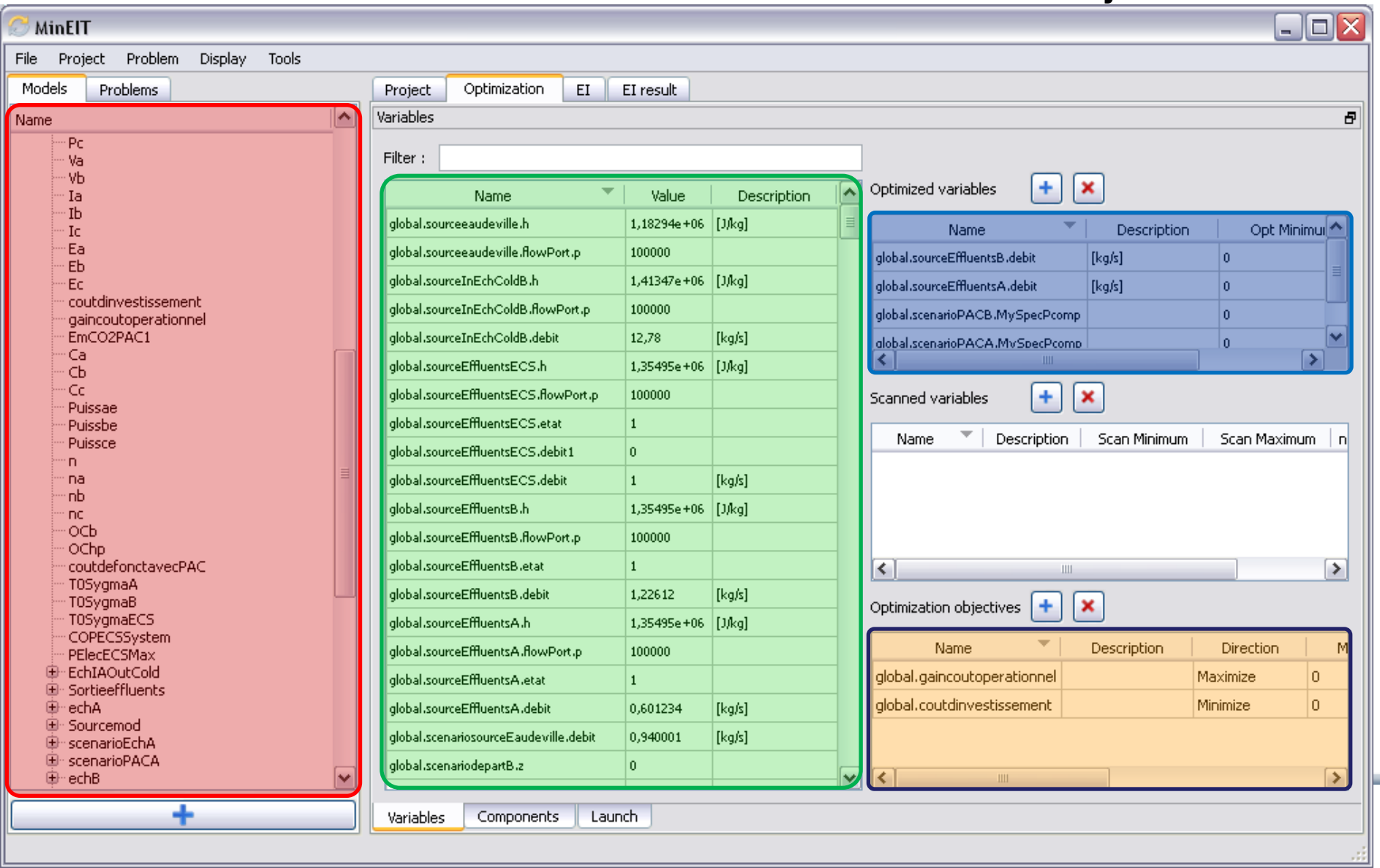
OMOptim – Optimization (1)

Model structure

Model Variables

Optimized parameters

Optimized Objectives



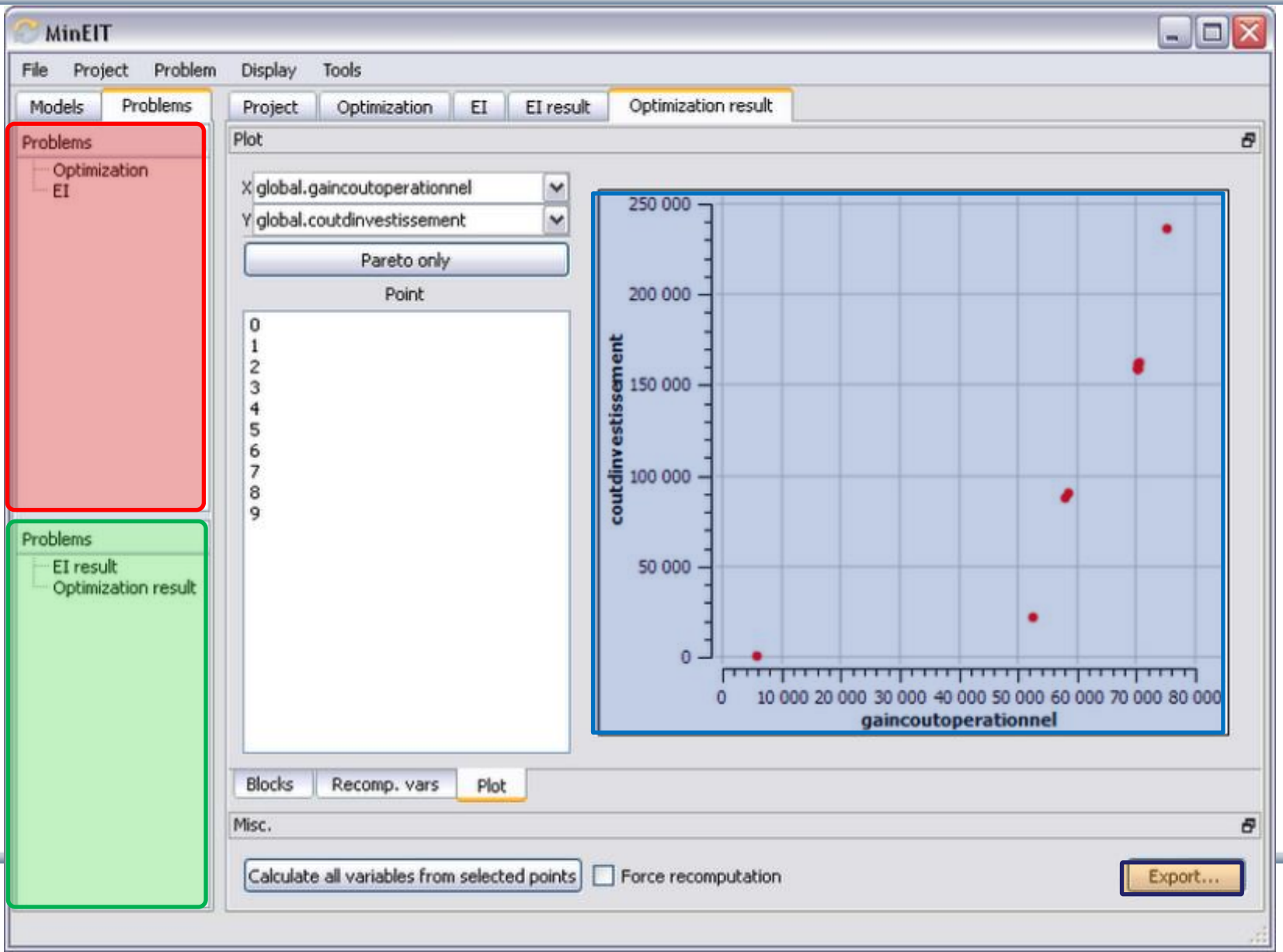
OMOptim – Optimization (2)

Problems

Solved problems

Result plot

Export result data .csv



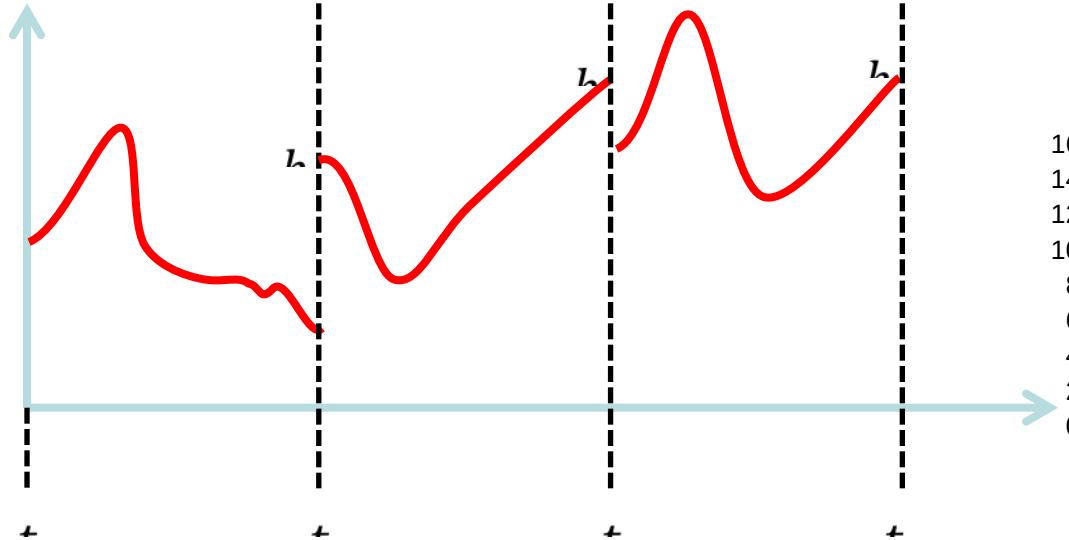
MODELICA

Multiple-Shooting and Collocation Dynamic Trajectory Optimization

- Minimize a goal function subject to model equation constraints, useful e.g. for NMPC
- Multiple Shooting/Collocation
  - Solve sub-problem in each sub-interval

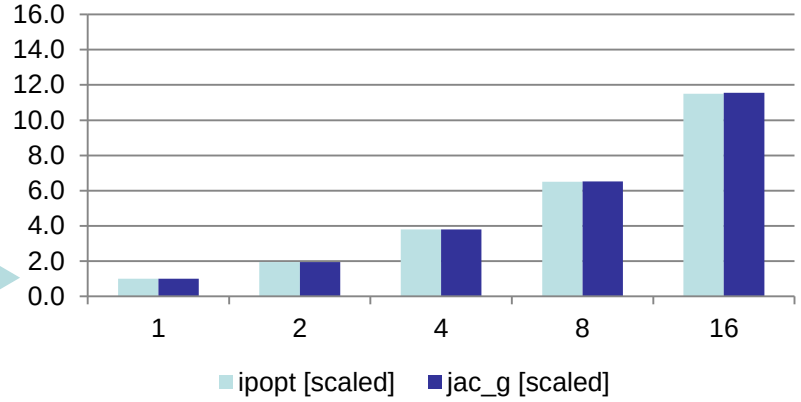
In OpenModelica 1.9.1 beta release Jan 2014.

$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t)\, dt \approx F(t_i, t_{i+1}, h_i, u_i), \qquad x_i(t_i) = h_i$$



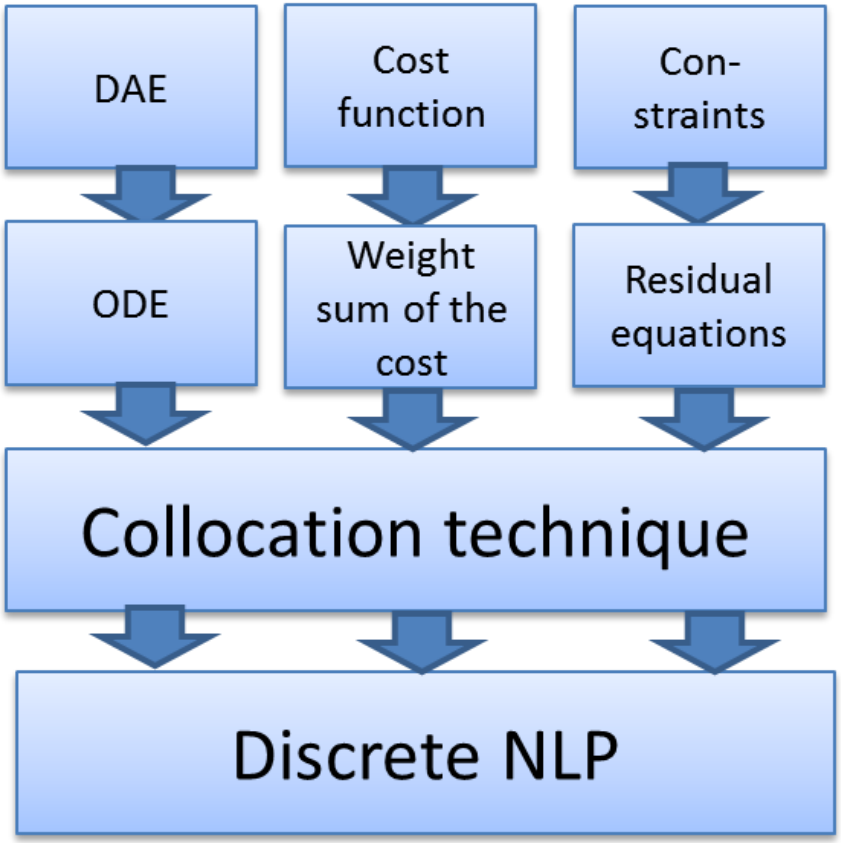
Example speedup, 16 cores:

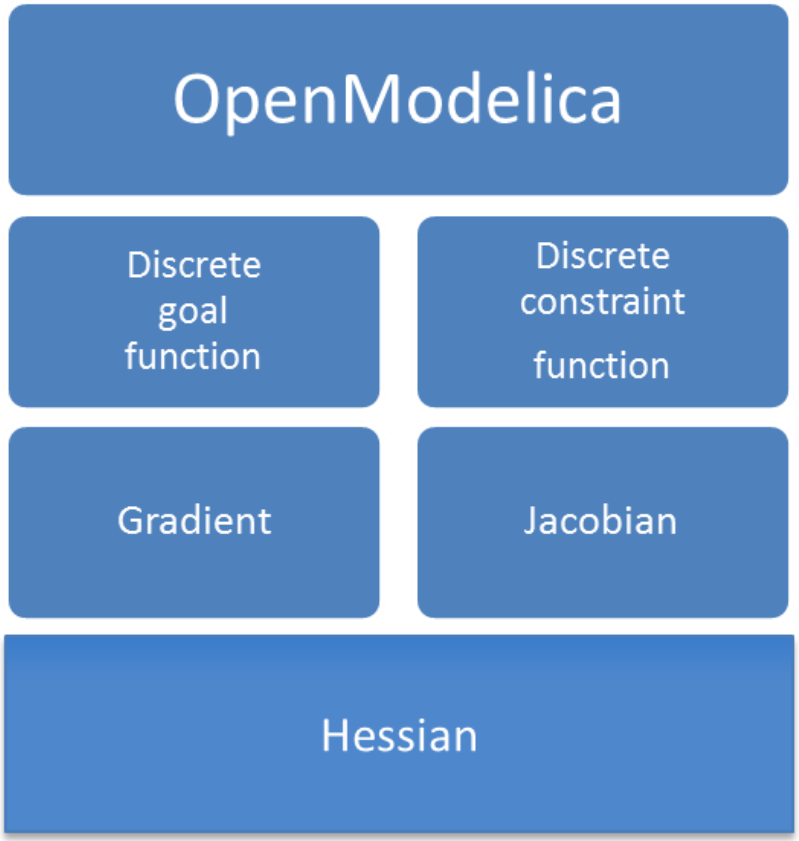
MULTIPLE_COLLOCATION



Copyright © Open Source Modelica Consortium

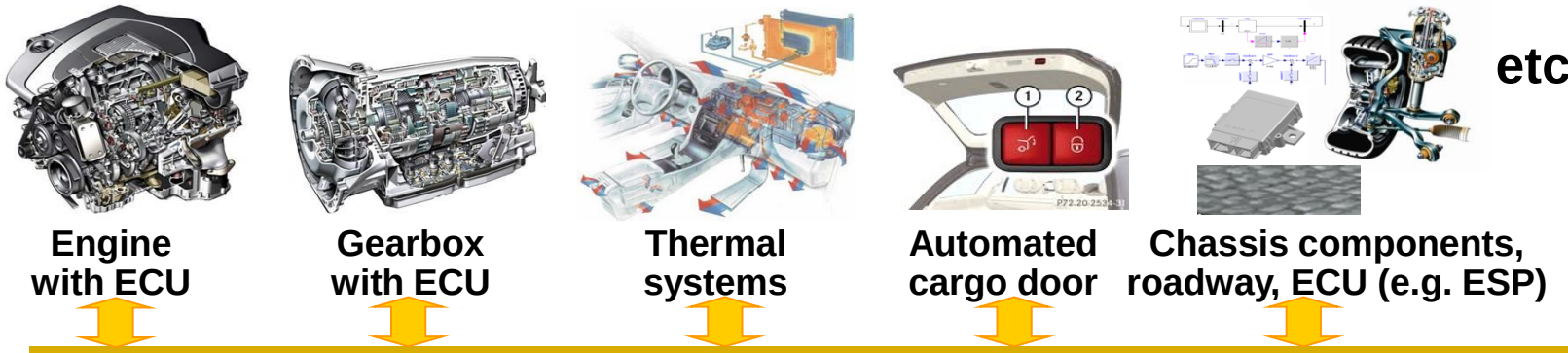
OpenModelica Dynamic Optimization Collocation





Copyright © Open Source Modelica Consortium

General Tool Interoperability & Model Exchange Functional Mock-up Interface (FMI)



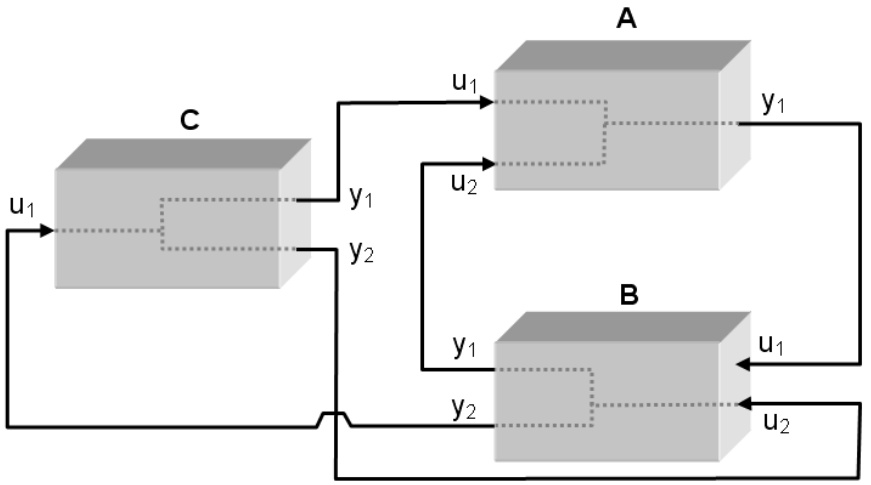
courtesy Daimler

- FMI development was started by ITEA2 MODELISAR project. FMI is a Modelica Association Project now
- **Version 1.0**
- FMI for Model Exchange (released Jan 26,2010)
- FMI for Co-Simulation (released Oct 12,2010)
- **Version 2.0**
- FMI for Model Exchange and Co-Simulation (released July 25,2014)
- **> 50 tools** supporting it (https://www.fmi-standard.org/tools)

Copyright © Open Source Modelica Consortium

Functional Mockup Units – FMUs

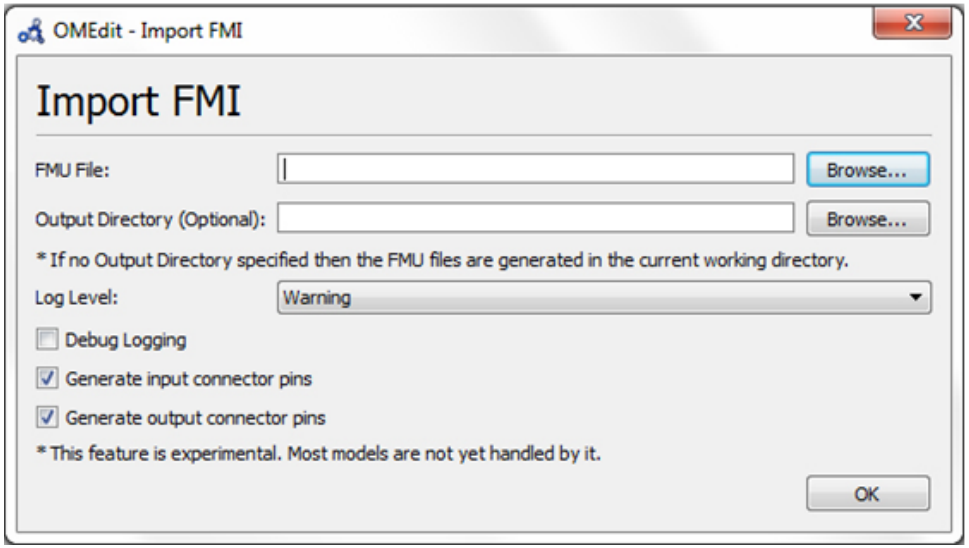
- Import and export of input/output blocks –
  Functional Mock-Up Units – FMUs, described by
  - differential-, algebraic-, discrete equations,
  - with time-, state, and step-events
- An FMU can be large (e.g. 100 000 variables)
- An FMU can be used in an embedded system (small overhead)
- FMUs can be connected together



Copyright © Open Source Modelica Consortium

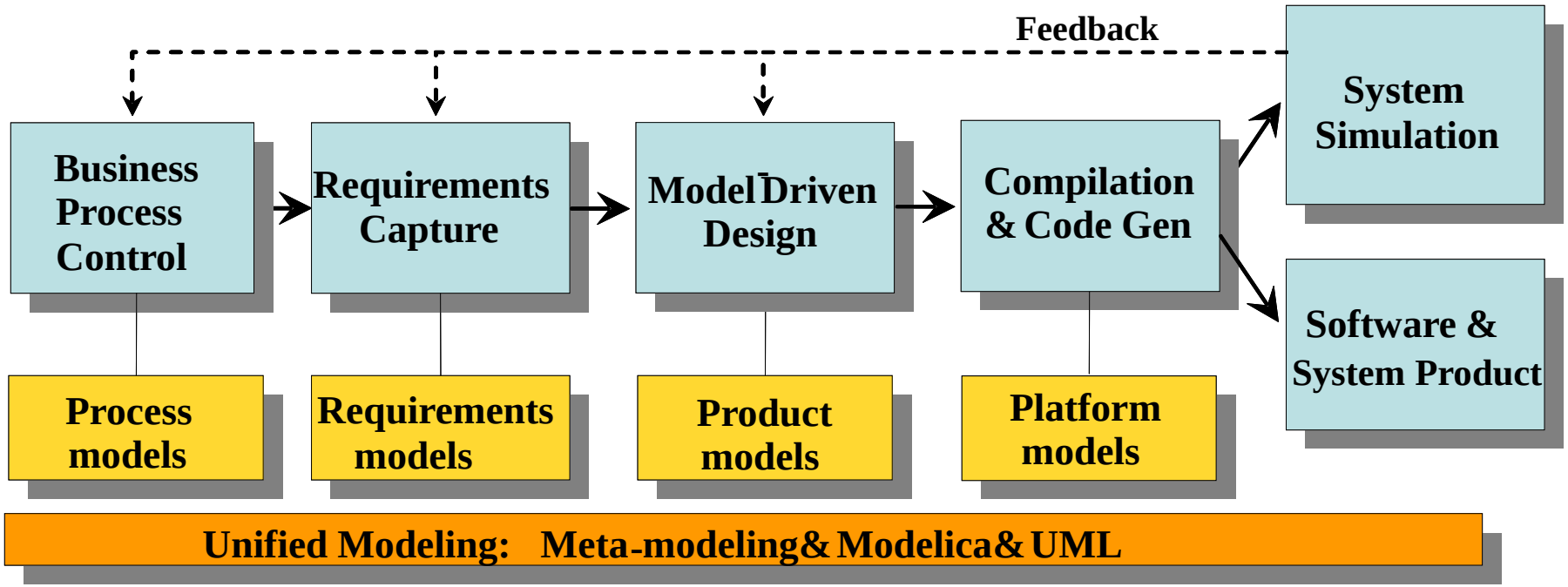
FMI in OpenModelica

- Model Exchange implemented (FMI 1.0 and FMI 2.0)
- FMI 2.0 Co-simulation is under development. A prototype of FMI 2.0 including tool co-simulation is available
- Ongoing work to support full FMI 2.0
- The FMI interface is accessible via the **OpenModelica scripting environment** and the **OpenModelica connection editor**


OMEdit - Import FMI
Import FMI
FMU File:
Browse...
Output Directory (Optional):
Browse...
* If no Output Directory specified then the FMU files are generated in the current working directory.
Log Level:
Warning
Debug Logging
Generate input connector pins
Generate output connector pins
* This feature is experimental. Most models are not yet handled by it.
OK

Copyright © Open Source Modelica Consortium

OPENPROD – Large 28-partner European Project, 2009-2012
Vision of Cyber-Physical Model-Based Product Development

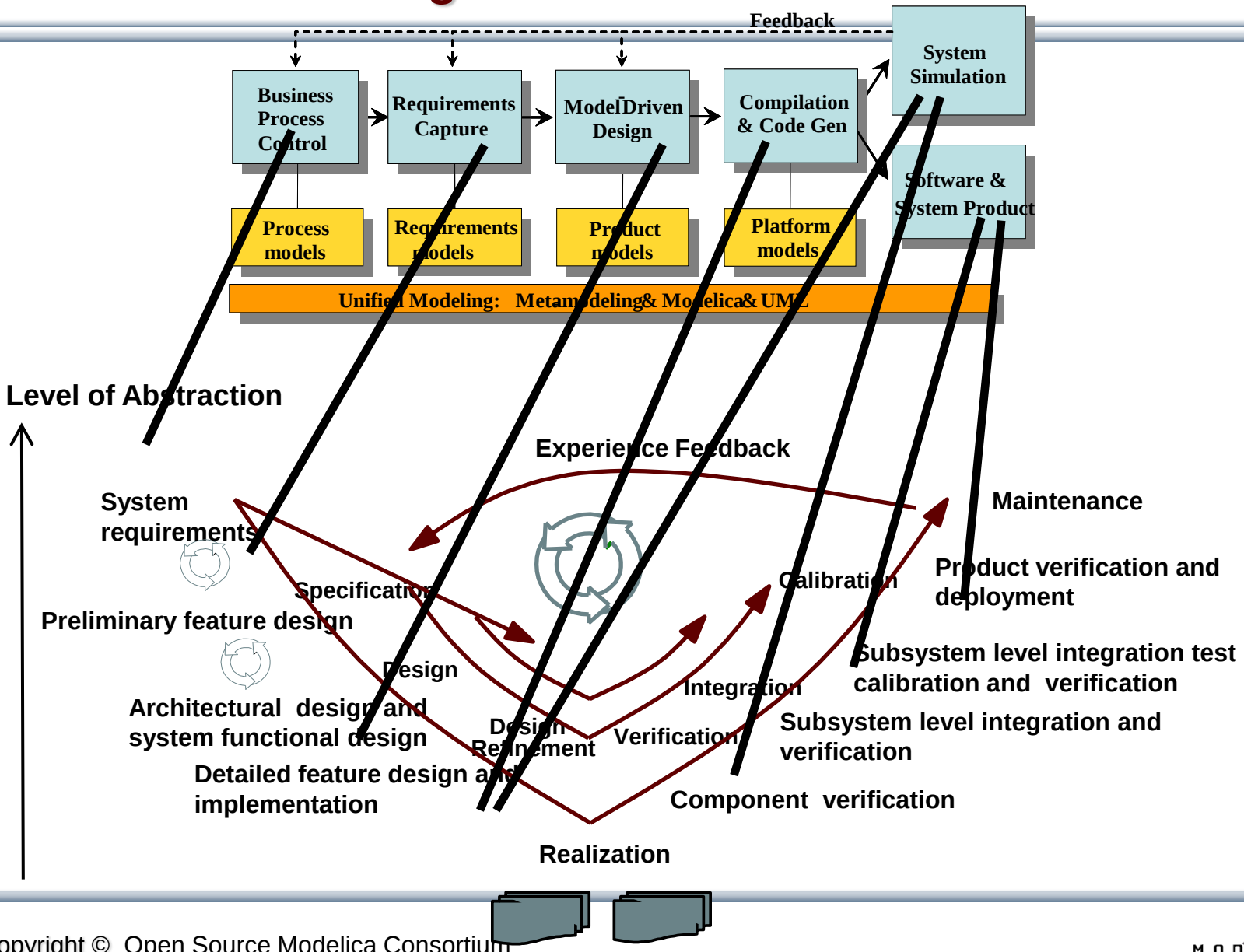


OPENPROD Vision of unified modeling framework for model-based product development.

Open Standards – Modelica (HW, SW) and UML (SW)

Copyright © Open Source Modelica Consortium

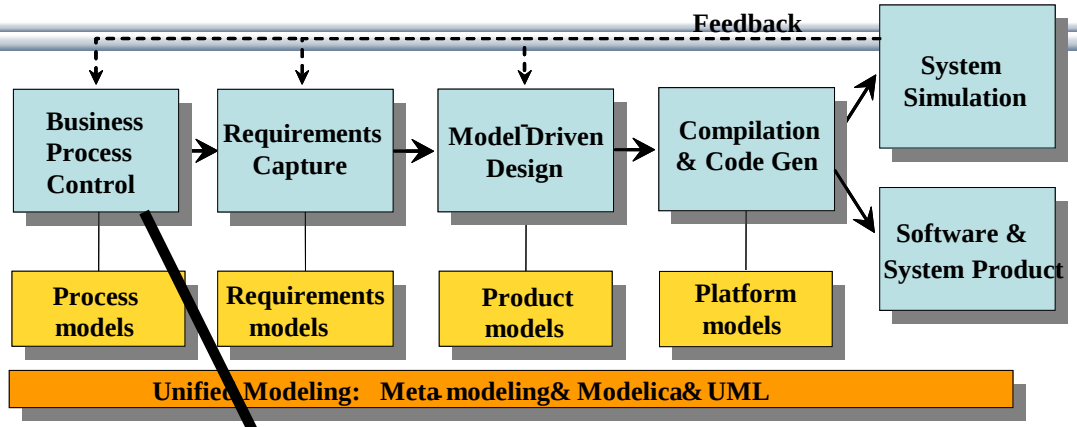
OPENPROD Model-Based Development Environment Covers Product-Design V



Copyright © Open Source Modelica Consortium

Documentation, Version and Configuration Management

Business Process Control and Modeling



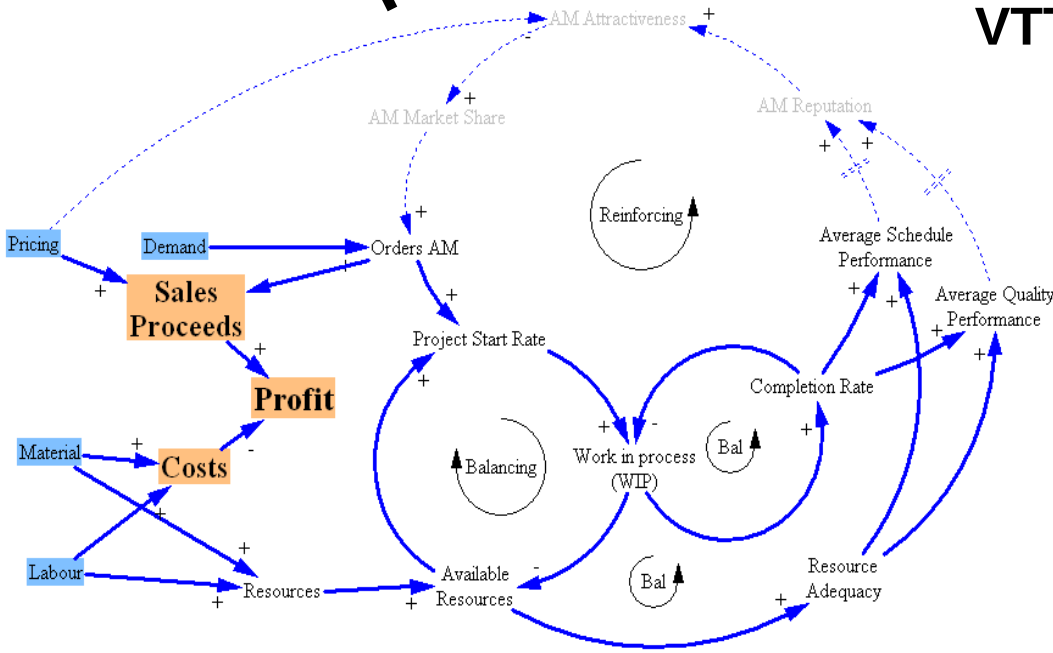
VTT Simantics
Business process modeler

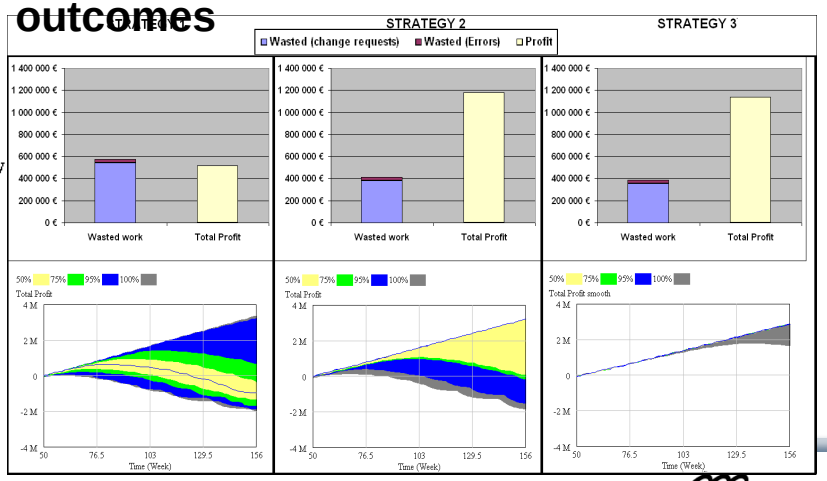
OpenModelica
compiler & simulator

OpenModelica based simulation

Metso Business model & simulation
VTT Simantics Graphic Modeling To

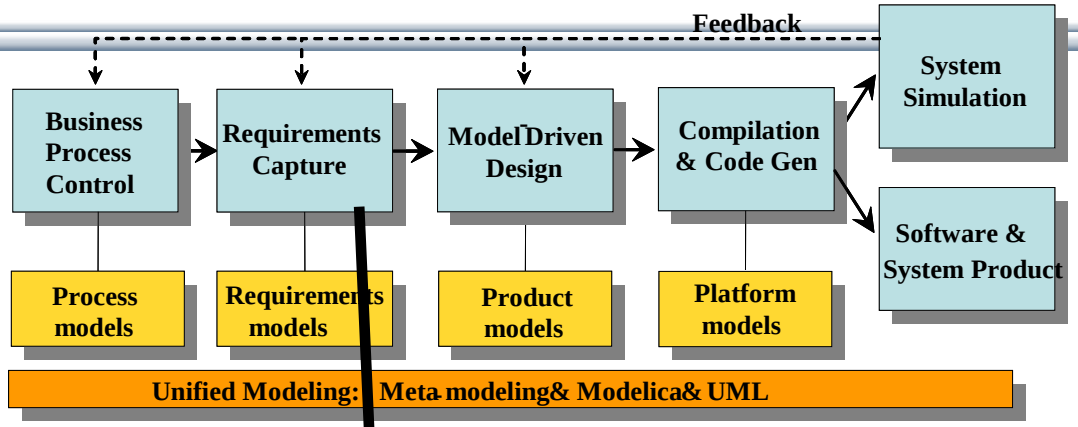
Simulation of 3 strategies with outcomes





MODELICA

Requirement Capture



OpenModelica based simulation

vVDR (virtual Verification of Designs against Requirements)

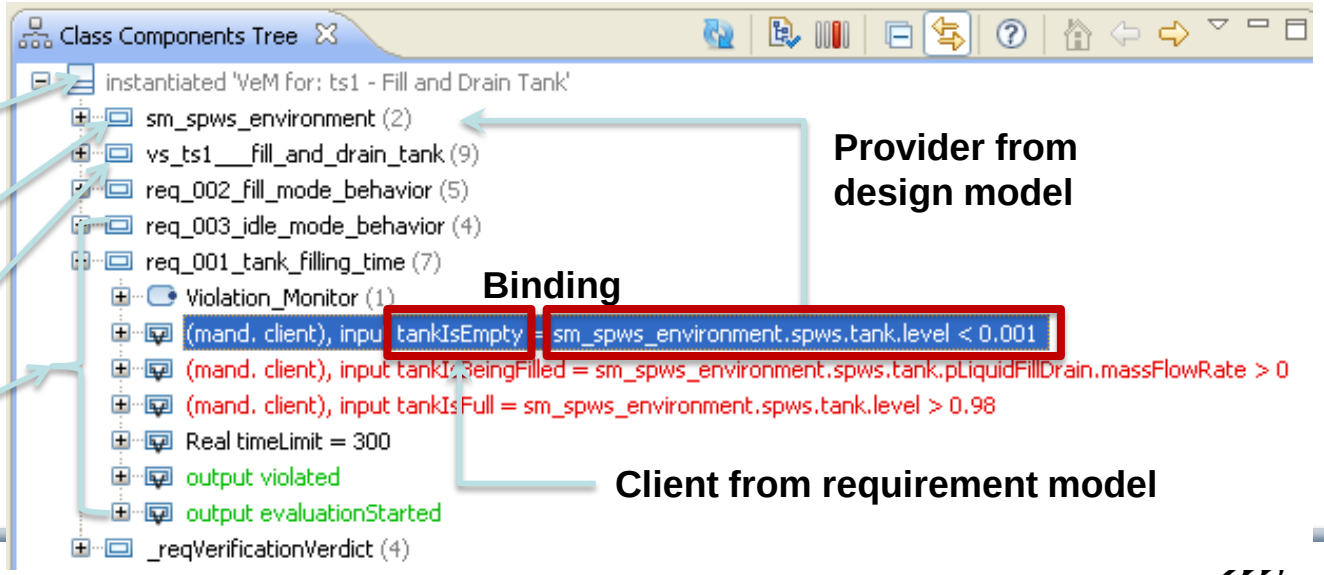
in ModelicaML UML/Modelica Profile, part of OpenModelica

Verification Model

Design Model

Scenario Model

Requirement Models



Provider from design model

Binding

Client from requirement model

Copyright © Open Source Modelica Consortium

OpenModelica – ModelicaML UML Profile
SysML/UML to Modelica OMG Standardization

- ModelicaML is a UML Profile for SW/HW modeling
  - Applicable to "pure" UML or to other UML profiles, e.g. SysML
- Standardized Mapping UML/SysML to Modelica
  - Defines transformation/mapping for **executable** models
  - Being **standardized** by OMG
- ModelicaML
  - Defines graphical concrete syntax (graphical notation for diagram) for representing Modelica constructs integrated with UML
  - Includes graphical formalisms (e.g. State Machines, Activities, Requirements)
    - Which do not exist in Modelica language
    - Which are translated into executable Modelica code
  - Is defined towards generation of executable Modelica code
  - Current implementation based on the Papyrus UML tool + OpenModelica

Copyright © Open Source Modelica Consortium

Example: Simulation and Requirements Evaluation

«model»
(TwoTanksSystemExample::SystemSimulations)
TankSystemSimulation

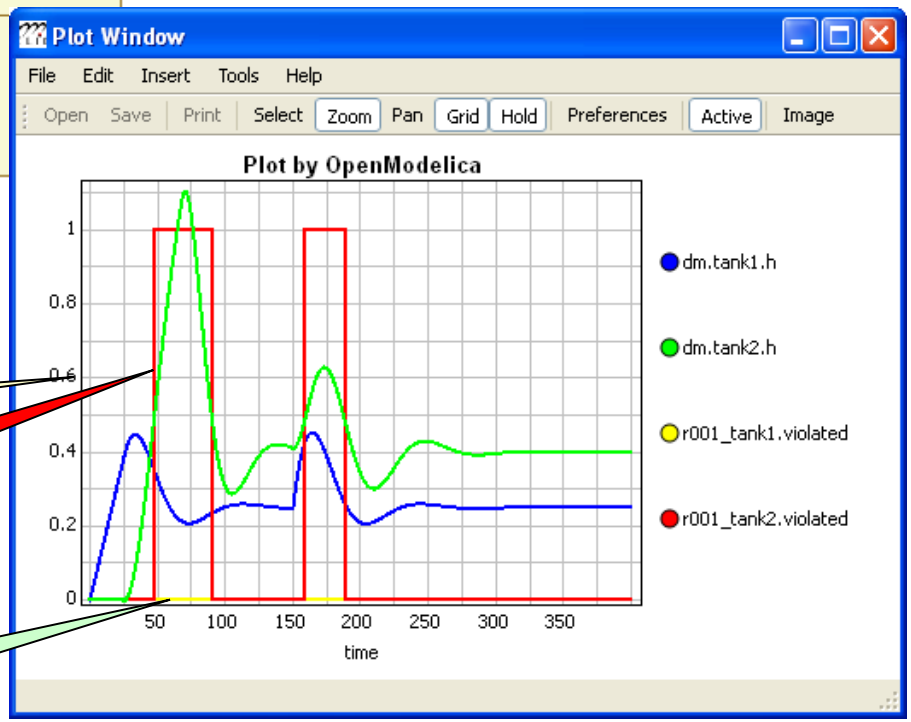
«component» dm: TanksConnectedPI
«requirementInstance» r001_tank1: Max level of liquid in a tank
«requirementInstance» r001_tank2: Max level of liquid in a tank
«requirementInstance» r002_tank1: Volume of the tank1

Req. 001 is instantiated 2 times (there are 2 tanks in the system)

tank-height is 0.6m

Req. 001 for the tank2 is violated

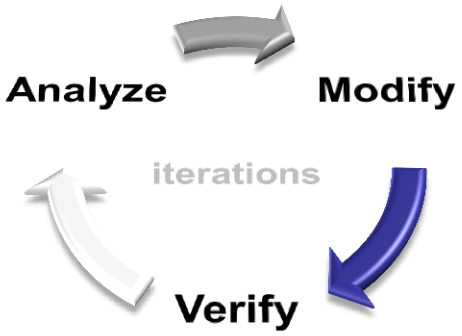
Req. 001 for the tank1 is not violated



Copyright © Open Source Modelica Consortium

vVDR Method – virtual Verification of Designs vs Requirements

Actor	Task	Created Artifact
	Formalize Requirements	RMM — Requirement Monitor Models
	Formalize Designs	DAM — Designs Alternative Models
	Formalize Scenarios	SM — Scenario Models
AUTOMATED	Create Verification Models *	VM — Verification Models
AUTOMATED	Execute and Create Report	Reports
	Analyze Results	

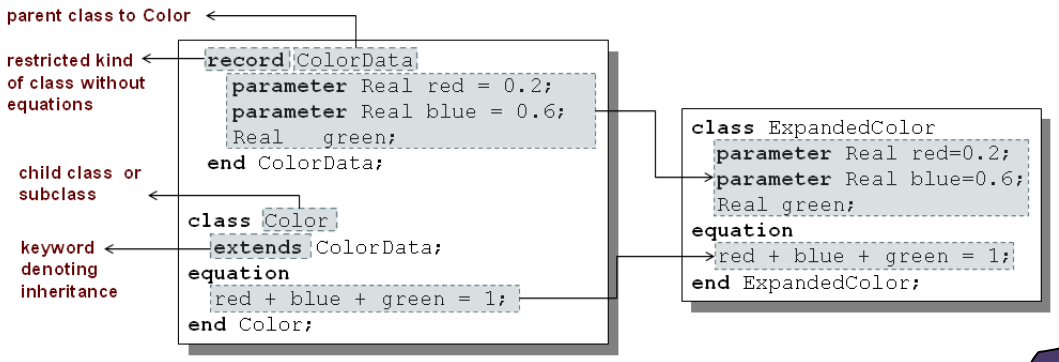


Goal: Enable on-demand verification of designs against requirements using automated model composition at any time during development.

Copyright © Open Source Modelica Consortium

Part III

Modelica language concepts and textual modeling



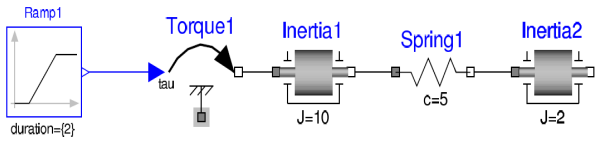
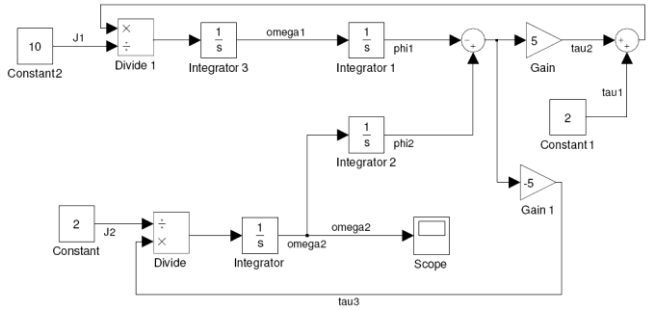
Typed Declarative Equation-based Textual Language

Hybrid Modeling

Copyright © Open Source Modelica Consortium

Acausal Modeling

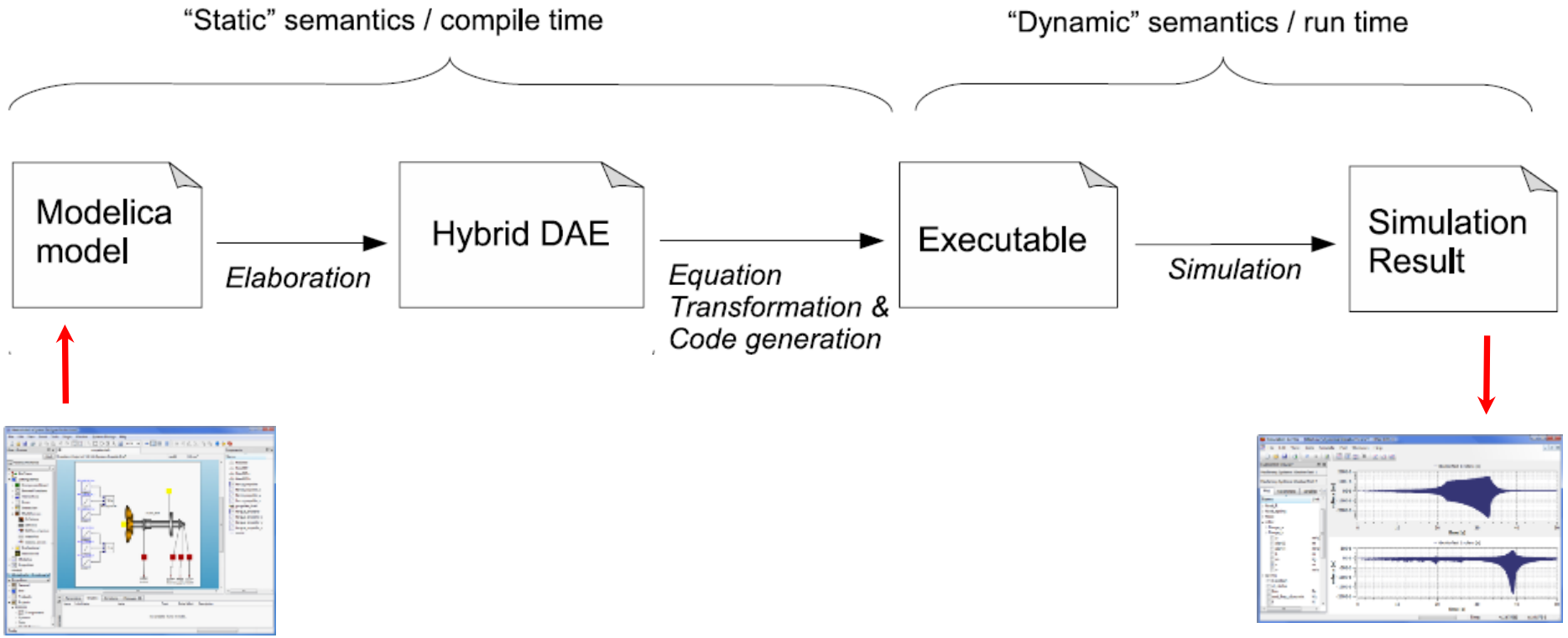
The order of computations is not decided at modeling time

	Acausal	Causal
Visual Component Level		

Equation Level	A resistor *equation*:
	`R*i = v;`
		Causal possibilities:
		`i := v/R;`
		`v := R*i;`
		`R := v/i;`

Copyright © Open Source Modelica Consortium

Typical Simulation Process



Copyright © Open Source Modelica Consortium

Simple model - Hello World!

Equation: x’ = - x
Initial condition: x(0) = 1

Name of model

Initial condition

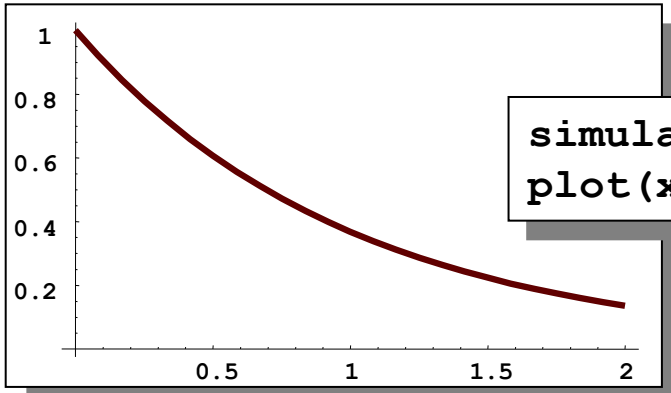
Continuous-time variable

Parameter, constant during simulation

Differential equation

```
model HelloWorld "A simple equation"
  Real x(start=1);
  parameter Real a = -1;
equation
  der(x)= a*x;
end HelloWorld;
```

Simulation in OpenModelica environment




```
simulate(HelloWorld, stopTime = 2)
plot(x)
```


Copyright © Open Source Modelica Consortium

Modelica Variables and Constants

- Built-in primitive data types

  Boolean **true** or **false**

  Integer Integer value, e.g. **42** or **–3**

  Real Floating point value, e.g. **2.4e-6**

  String String, e.g. **“Hello world”**

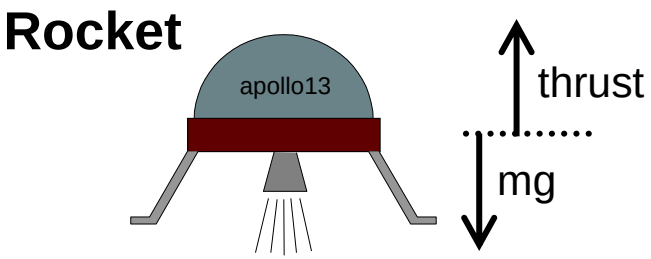
  Enumeration Enumeration literal e.g. **ShirtSize.Medium**

- Parameters are constant during simulation
- Two types of constants in Modelica
  - **`constant`**
  - **`parameter`**

```
constant  Real    PI=3.141592653589793;
constant  String  redcolor = "red";
constant  Integer one = 1;
parameter Real    mass = 22.5;
```


Copyright © Open Source Modelica Consortium

A Simple Rocket Model



$$acceleration = \frac{thrust - mass \cdot gravity}{mass}$$

$$mass' = -massLossRate \cdot \mathrm{abs}(thrust)$$

$$altitude' = velocity$$

$$velocity' = acceleration$$

```
class Rocket "rocket class"
  parameter String name;
  Real mass(start=1038.358);
  Real altitude(start= 59404);
  Real velocity(start= -2003);
  Real acceleration;
  Real thrust;  // Thrust force on rocket
  Real gravity; // Gravity forcefield
  parameter Real massLossRate=0.000277;
equation
  (thrust-mass*gravity)/mass = acceleration;
  der(mass)  = -massLossRate * abs(thrust);
  der(altitude) = velocity;
  der(velocity) = acceleration;
end Rocket;
```

new model

parameters (changeable before the simulation)

floating point type

differentiation with regards to time

declaration comment

start value

name + default value

mathematical equation (acausal)

Copyright © Open Source Modelica Consortium

Celestial Body Class

A class declaration creates a *type name* in Modelica

```
class CelestialBody
  constant  Real    g = 6.672e-11;
  parameter Real    radius;
  parameter String  name;
  parameter Real    mass;
end CelestialBody;
```

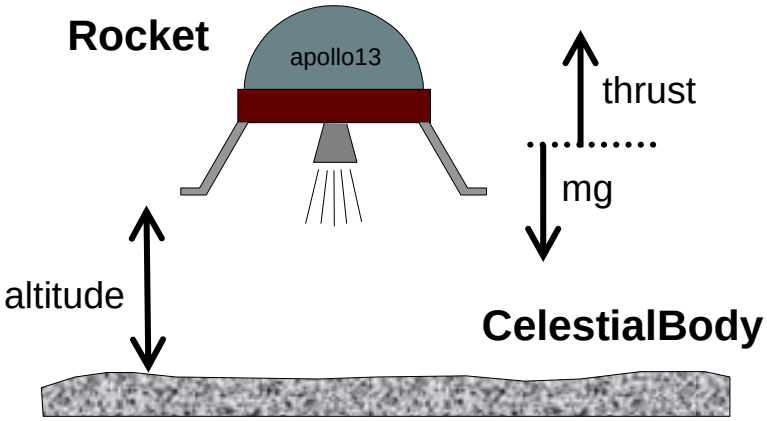
An *instance* of the class can be declared by *prefixing* the type name to a variable name

```
...
CelestialBody moon;
...
```

The declaration states that **`moon`** is a variable containing an object of type **`CelestialBody`**

Copyright © Open Source Modelica Consortium

Moon Landing



$$apollo.gravity = \frac{moon.g \cdot moon.mass}{(apollo.altitude + moon.radius)^2}$$

only access inside the class

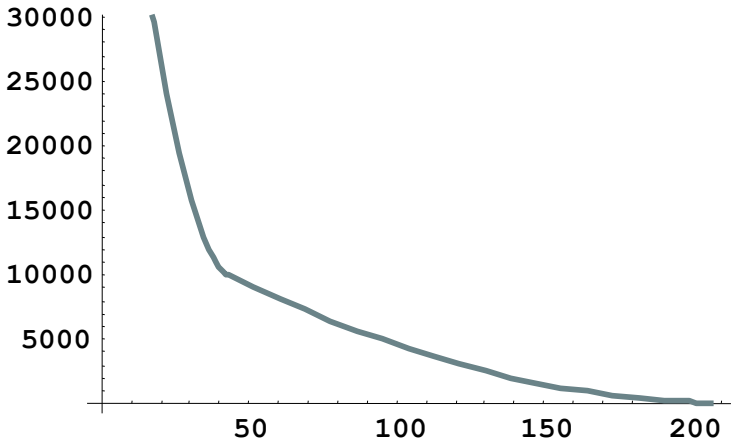
access by dot notation outside the class

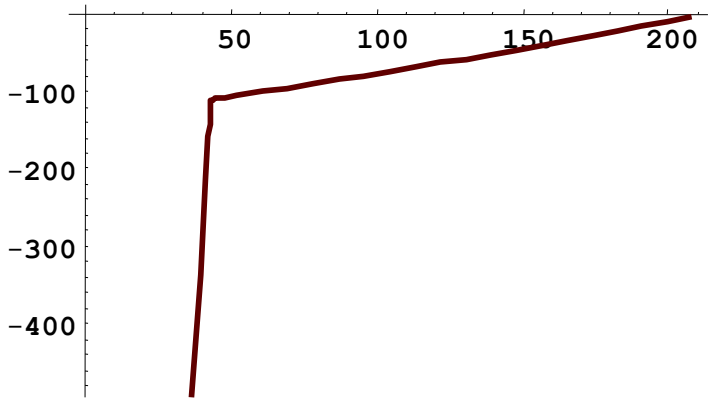
```
class MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
protected
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
public
  Rocket         apollo(name="apollo13");
  CelestialBody   moon(name="moon",mass=7.382e22,radius=1.738e6);
equation
  apollo.thrust = if (time < thrustDecreaseTime) then force1
                  else if (time < thrustEndTime) then force2
                  else 0;
  apollo.gravity=moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end MoonLanding;
```


Copyright © Open Source Modelica Consortium

Simulation of Moon Landing

```
simulate(MoonLanding, stopTime=230)
plot(apollo.altitude, xrange={0,208})
plot(apollo.velocity, xrange={0,208})
```





It starts at an altitude of 59404 (not shown in the diagram) at time zero, gradually reducing it until touchdown at the lunar surface when the altitude is zero

The rocket initially has a high negative velocity when approaching the lunar surface. This is reduced to zero at touchdown, giving a smooth landing

Copyright © Open Source Modelica Consortium

Specialized Class Keywords

- Classes can also be declared with other keywords, e.g.: `model`, `record`, `block`, `connector`, `function`, ...
- Classes declared with such keywords have specialized properties
- Restrictions and enhancements apply to contents of specialized classes
- After Modelica 3.0 the `class` keyword means the same as `model`

- Example: (Modelica 2.2). A `model` is a class that cannot be used as a connector class
- Example: A `record` is a class that only contains data, with no equations
- Example: A `block` is a class with fixed input-output causality

```
model CelestialBody
  constant  Real    g = 6.672e-11;
  parameter Real    radius;
  parameter String  name;
  parameter Real    mass;
end CelestialBody;
```


Copyright © Open Source Modelica Consortium

Modelica Functions

- Modelica Functions can be viewed as a specialized class with some restrictions and extensions
- A function can be called with arguments, and is instantiated dynamically when called
- := means assignment

```
function sum
  input  Real arg1;
  input  Real arg2;
  output Real result;
algorithm
  result := arg1+arg2;
end sum;
```


Copyright © Open Source Modelica Consortium

Function Call – Example Function with for-loop

Example Modelica function call:

```
...
  p = polynomialEvaluator({1,2,3,4},21)
```

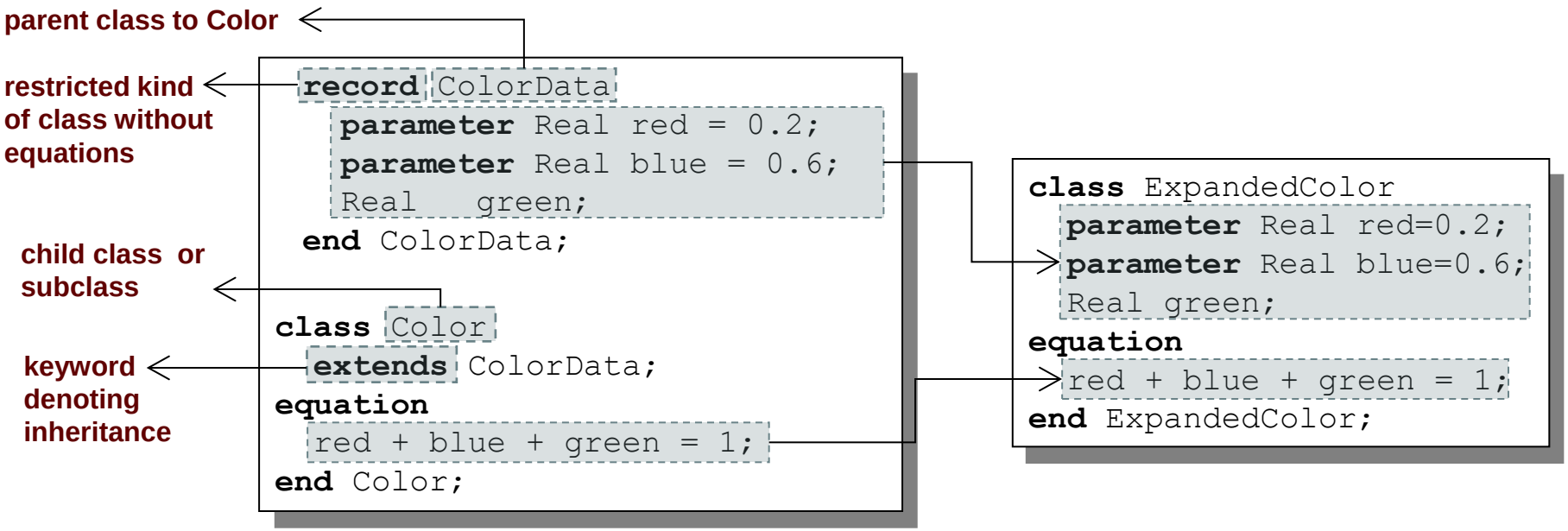
```
function PolynomialEvaluator
  input  Real A[:];    // array, size defined
                       // at function call time
  input  Real x := 1.0;// default value 1.0 for x
  output Real sum;
protected
  Real   xpower;          // local variable xpower
algorithm
  sum := 0;
  xpower := 1;
  for i in 1:size(A,1) loop
    sum := sum + A[i]*xpower;
    xpower := xpower*x;
  end for;
end PolynomialEvaluator;
```

`{1,2,3,4}` becomes the value of the coefficient vector `A`, and `21` becomes the value of the formal parameter `x`.

The function `PolynomialEvaluator` computes the value of a polynomial given two arguments:
a coefficient vector `A` and a value of `x`.

Copyright © Open Source Modelica Consortium

Inheritance

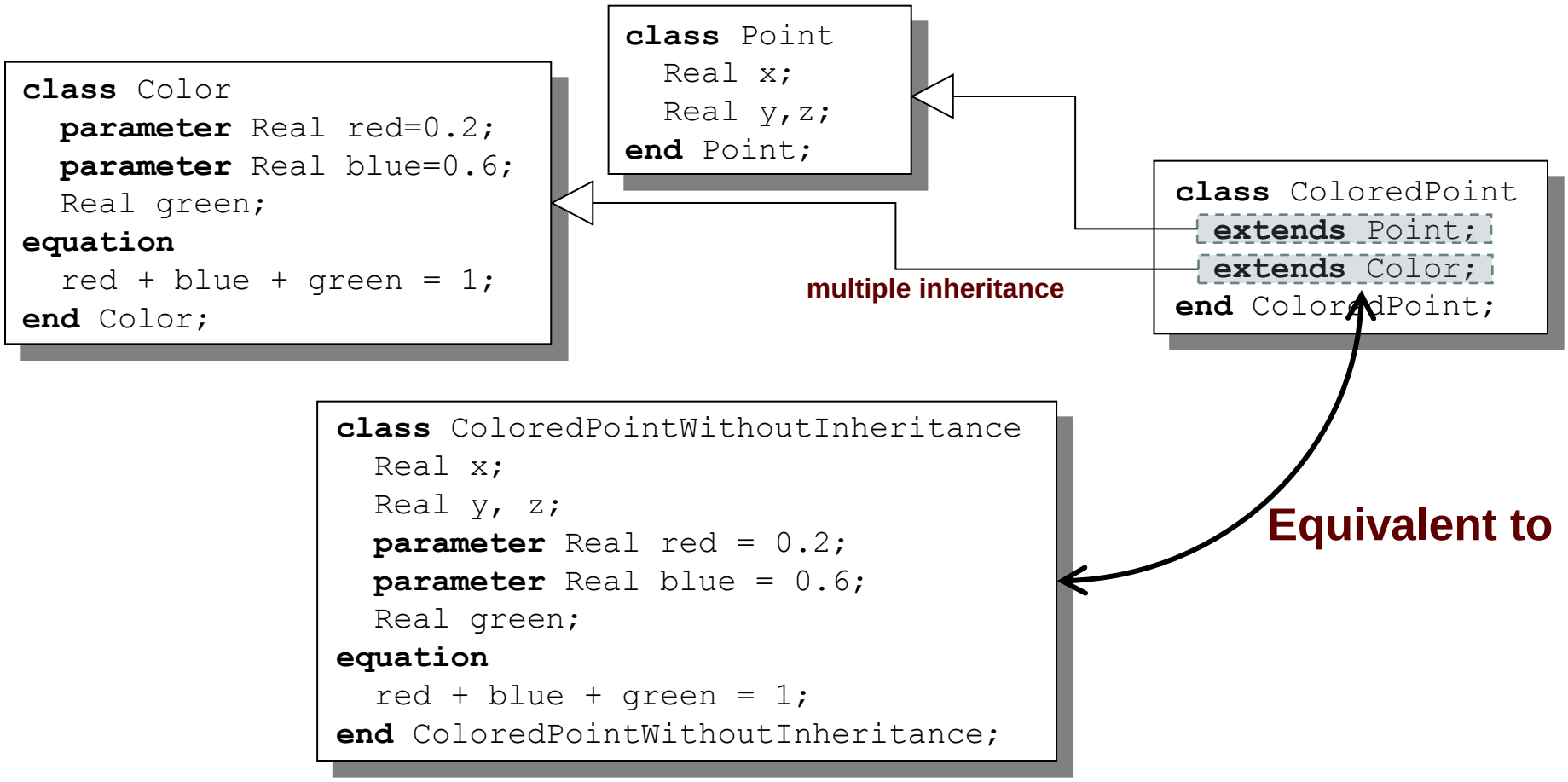


Data and behavior: field declarations, equations, and certain other contents are *copied* into the subclass

Copyright © Open Source Modelica Consortium

Multiple Inheritance

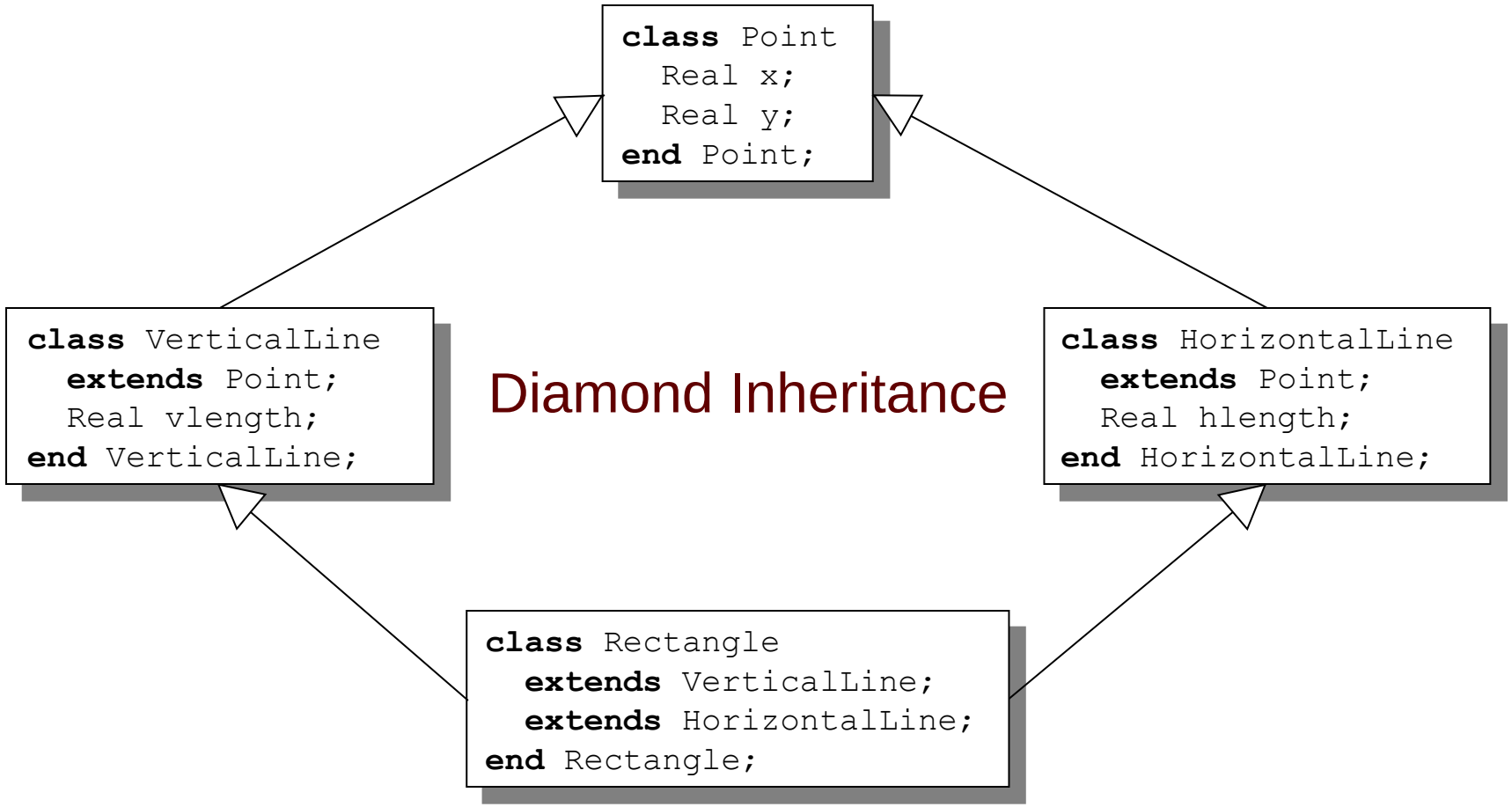
Multiple Inheritance is fine – inheriting both geometry and color



Copyright © Open Source Modelica Consortium

Multiple Inheritance cont'

Only one copy of multiply inherited class `Point` is kept



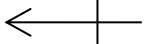
Copyright © Open Source Modelica Consortium

Simple Class Definition

- Simple Class Definition
  - Shorthand Case of Inheritance
- Example:

```
class SameColor = Color;
```

Equivalent to:

inheritance 

```
class SameColor
  extends Color;
end SameColor;
```

- Often used for introducing new names of types:

```
type Resistor = Real;
```

```
connector MyPin = Pin;
```


Copyright © Open Source Modelica Consortium

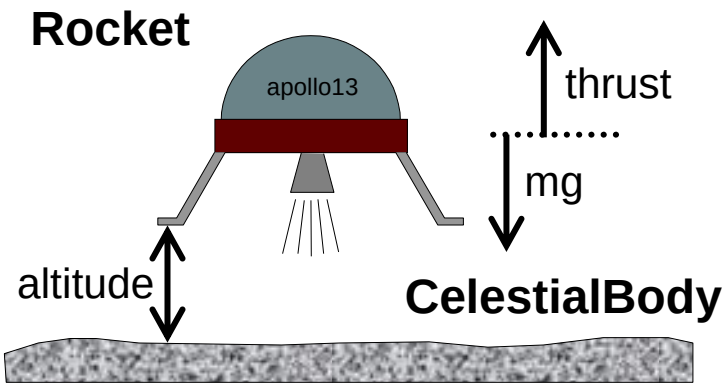
Inheritance Through Modification

- Modification is a concise way of combining inheritance with declaration of classes or instances
- A *modifier* modifies a declaration equation in the inherited class
- Example: The class `Real` is inherited, modified with a different `start` value equation, and instantiated as an `altitude` variable:

```
...
 Real altitude(start= 59404);
...
```


Copyright © Open Source Modelica Consortium

The Moon Landing - Example Using Inheritance (I)



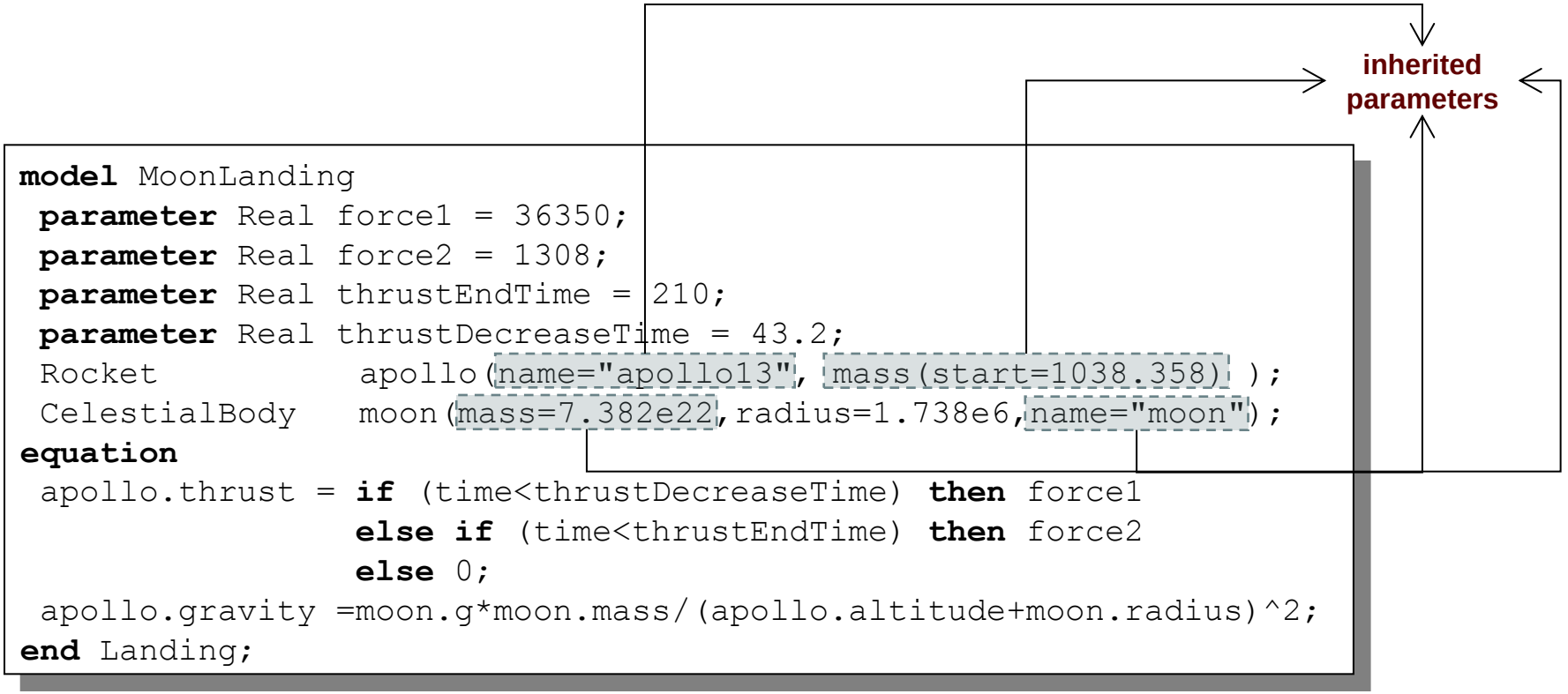

```
model Body "generic body"
  Real   mass;
  String name;
end Body;
```

```
model CelestialBody
  extends Body;
  constant  Real g = 6.672e-11;
  parameter Real radius;
end CelestialBody;
```

```
model Rocket "generic rocket class"
  extends Body;
  parameter Real massLossRate=0.000277;
  Real altitude(start= 59404);
  Real velocity(start= -2003);
  Real acceleration;
  Real thrust;
  Real gravity;
equation
  thrust-mass*gravity= mass*acceleration;
  der(mass)= -massLossRate*abs(thrust);
  der(altitude)= velocity;
  der(velocity)= acceleration;
end Rocket;
```


Copyright © Open Source Modelica Consortium

The Moon Landing - Example using Inheritance (II)

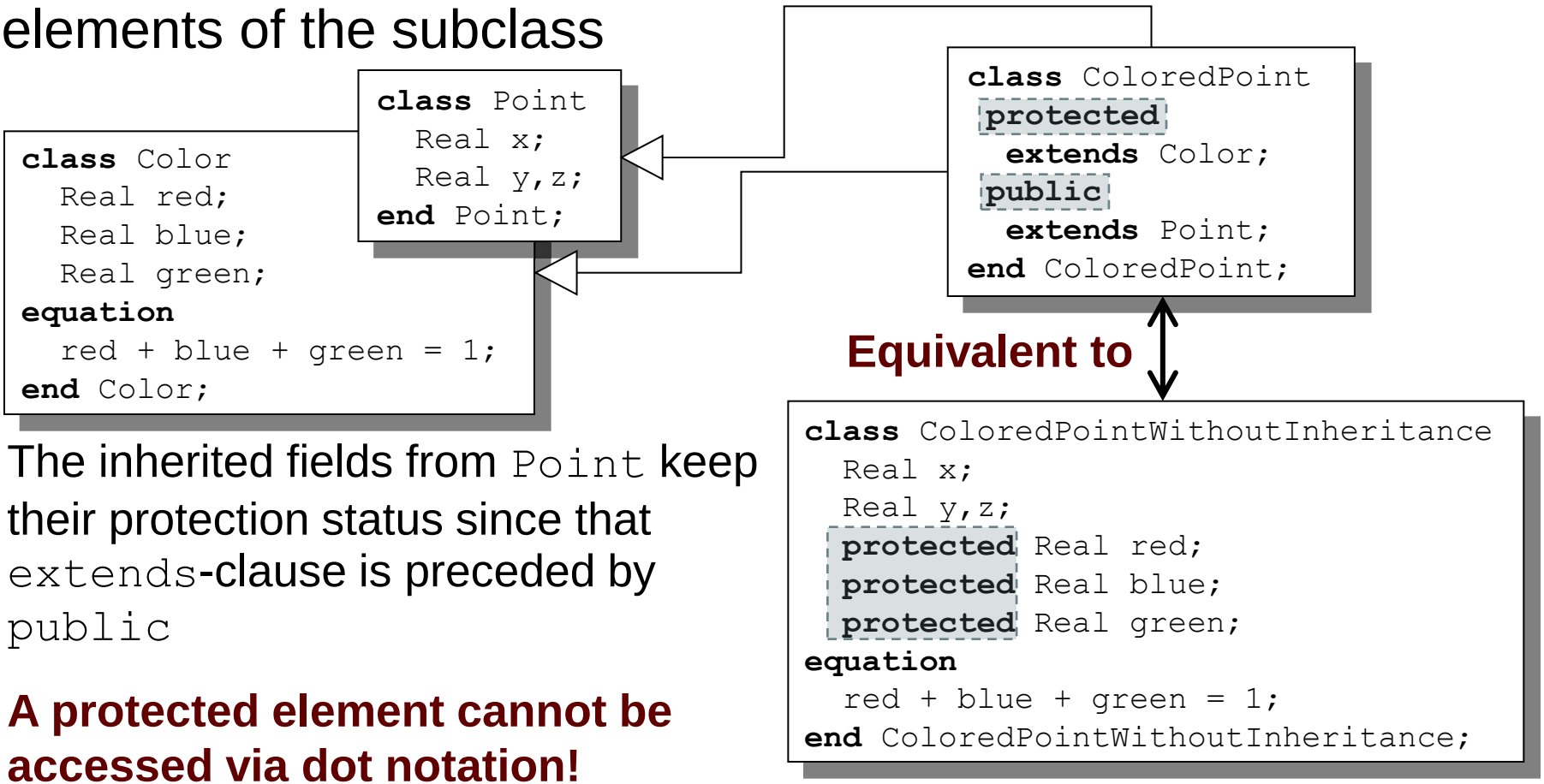



```
model MoonLanding
 parameter Real force1 = 36350;
 parameter Real force2 = 1308;
 parameter Real thrustEndTime = 210;
 parameter Real thrustDecreaseTime = 43.2;
 Rocket          apollo(name="apollo13", mass(start=1038.358) );
 CelestialBody   moon(mass=7.382e22,radius=1.738e6,name="moon");
equation
 apollo.thrust = if (time<thrustDecreaseTime) then force1
                 else if (time<thrustEndTime) then force2
                 else 0;
 apollo.gravity =moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;
```


Copyright © Open Source Modelica Consortium

Inheritance of Protected Elements

If an `extends`-clause is preceded by the `protected` keyword, all inherited elements from the superclass become protected elements of the subclass



The inherited fields from `Point` keep their protection status since that `extends`-clause is preceded by `public`

A protected element cannot be accessed via dot notation!

Copyright © Open Source Modelica Consortium

Exercises Part II
(30 minutes)

Copyright © Open Source Modelica Consortium

Exercises Part II

- Start OMNotebook (part of OpenModelica)
  - **Start-**>Programs->OpenModelica->OMNotebook
  - **Open File**: Exercises-ModelicaTutorial.onb from the directory you copied your tutorial files to.
  - **Note**: The DrModelica electronic book has been automatically opened when you started OMNotebook.
- Open Exercises-ModelicaTutorial.pdf (also available in printed handouts)

Copyright © Open Source Modelica Consortium

Exercises 2.1 and 2.2 (See also next two pages)

- Open the **Exercises-ModelicaTutorial.onb** found in the Tutorial directory you copied at installation.
- **Exercise 2.1**. Simulate and plot the HelloWorld example. Do a slight change in the model, re-simulate and re-plot. Try command-completion, val( ), etc.

```
class HelloWorld "A simple equation"
  Real x(start=1);
equation
  der(x)= -x;
end HelloWorld;
```

```
simulate(HelloWorld, stopTime = 2)
plot(x)
```

- Locate the VanDerPol model in DrModelica (link from Section 2.1), using OMNotebook!
- **Exercise 2.2**: Simulate and plot VanDerPol. Do a slight change in the model, re-simulate and re-plot.

Copyright © Open Source Modelica Consortium

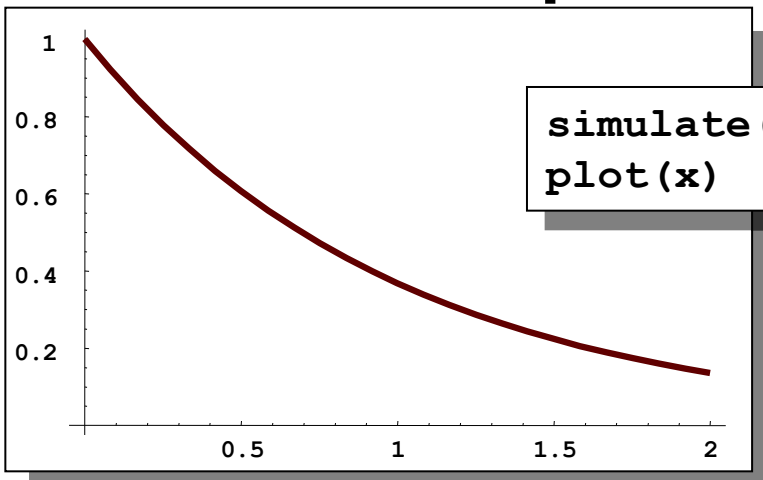
Exercise 2.1 – Hello World!

A Modelica "Hello World" model

Equation: x' = - x
Initial condition: x(0) = 1

```
class HelloWorld "A simple equation"
  parameter Real a=-1;
  Real x(start=1);
equation
  der(x)= a*x;
end HelloWorld;
```

Simulation in OpenModelica environment



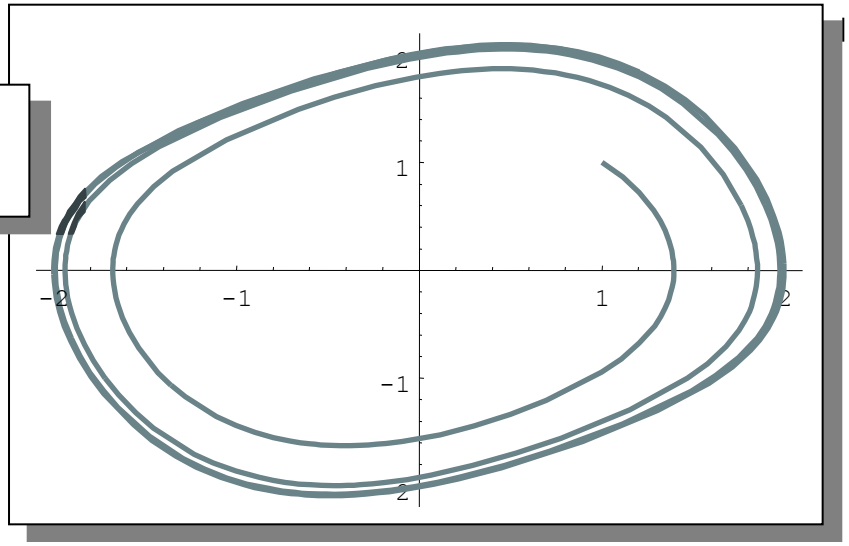

```
simulate(HelloWorld, stopTime = 2)
plot(x)
```


Copyright © Open Source Modelica Consortium

Exercise 2.2 – Van der Pol Oscillator

```
class VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x"; // x starts at 1
  Real y(start = 1)  "y coordinate";             // y starts at 1
  parameter Real lambda = 0.3;
equation
  der(x) = y;                              // This is the 1st diff equation //
  der(y) = -x + lambda*(1 - x*x)*y;  /* This is the 2nd diff equation */
end VanDerPol;
```

```
simulate(VanDerPol,stopTime = 25)
plotParametric(x,y)
```



Copyright © Open Source Modelica Consortium

Exercise 2.3 – DAE Example

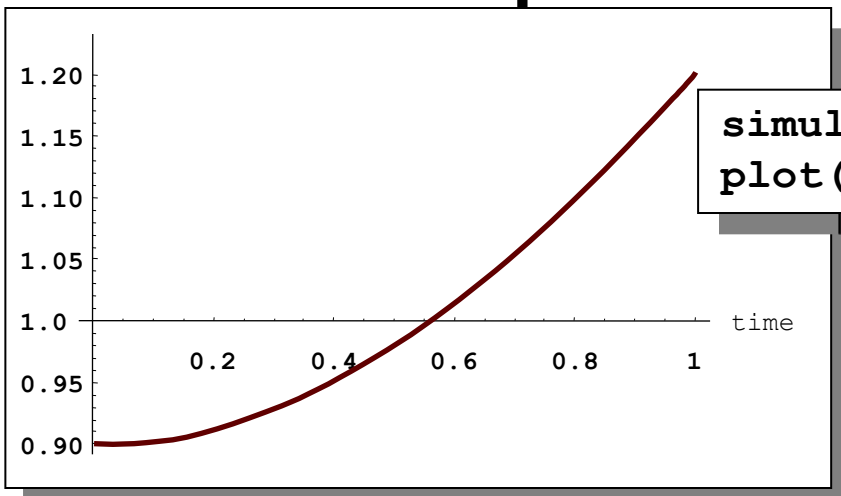
Include algebraic equation

Algebraic equations contain no derivatives

Exercise: Locate in DrModelica. Simulate and plot. Change the model, simulate+plot.

```
class DAEexample
  Real x(start=0.9);
  Real y;
equation
  der(y)+(1+0.5*sin(y))*der(x)
    = sin(time);
  x - y = exp(-0.9*x)*cos(y);
end DAEexample;
```

Simulation in OpenModelica environment




```
simulate(DAEexample, stopTime = 1)
plot(x)
```


Copyright © Open Source Modelica Consortium

Exercise 2.4 – Model the system below

- Model this Simple System of Equations in Modelica

$$\dot{x} = 2 * x * y - 3 * x$$

$$\dot{y} = 5 * y - 7 * x * y$$

$$x(0) = 2$$

$$y(0) = 3$$

Copyright © Open Source Modelica Consortium

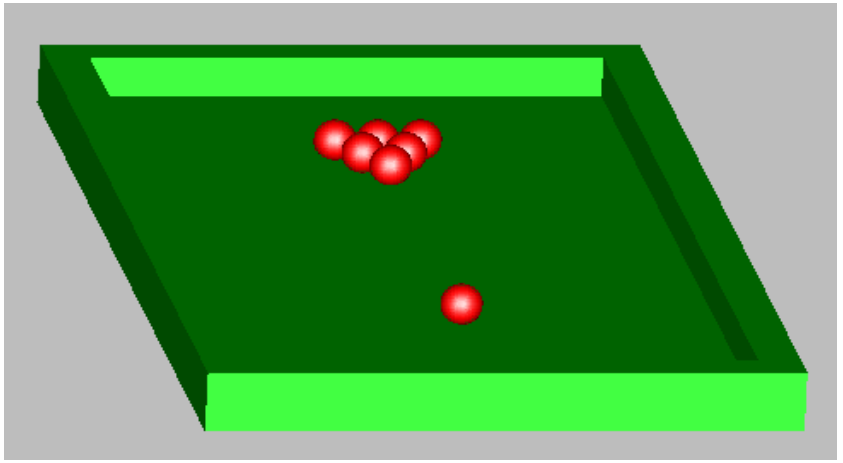
Exercise 2.5 – Functions

- a) Write a function, **`sum2`**, which calculates the sum of Real numbers, for a vector of arbitrary size.

- b) Write a function, **`average`**, which calculates the average of Real numbers, in a vector of arbitrary size. The function **`average`** should make use of a function call to **`sum2`**.

Copyright © Open Source Modelica Consortium

Discrete Events and Hybrid Systems

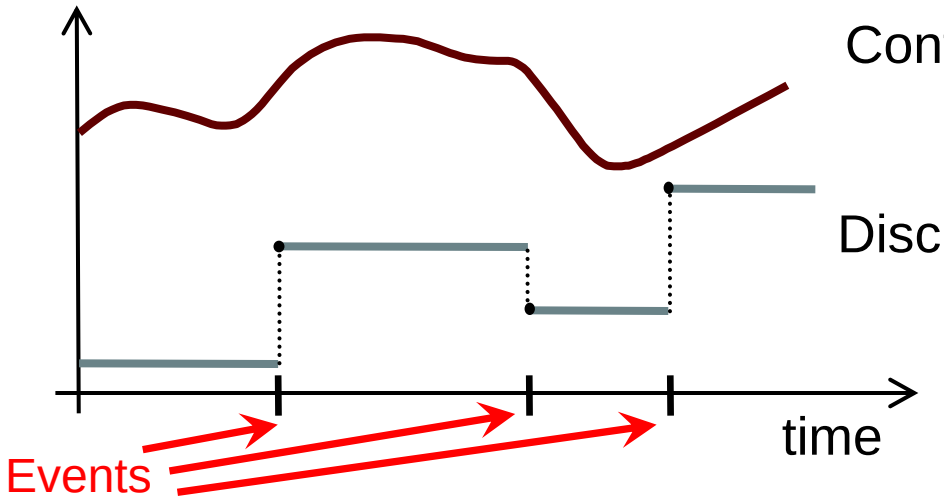


Picture: Courtesy Hilding Elmqvist

Copyright © Open Source Modelica Consortium

Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling




```
Real x;
Voltage v;
Current i;
```

```
discrete Real x;
Integer i;
Boolean b;
```

- A *point* in time that is instantaneous, i.e., has zero duration
- An *event condition* so that the event can take place
- A set of *variables* that are associated with the event
- Some *behavior* associated with the event,
  e.g. *conditional equations* that become active or are deactivated at the event

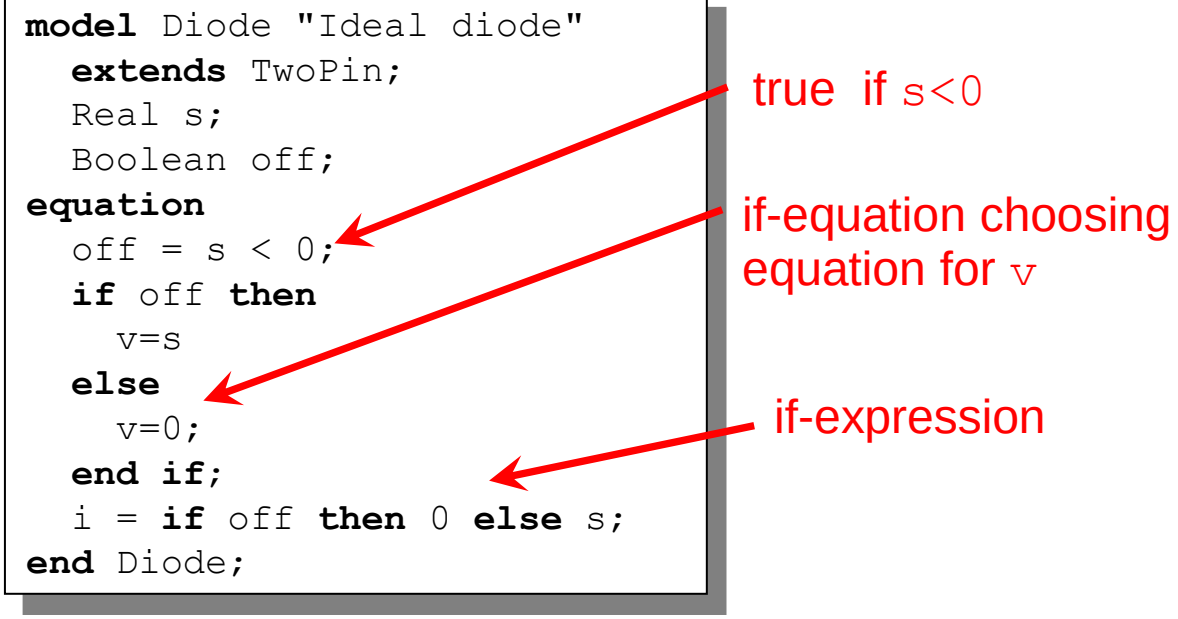
Copyright © Open Source Modelica Consortium

Event Creation – if

if-equations, if-statements, and if-expressions

if <*condition*> **then**
  <*equations*>
elseif <*condition*> **then**
  <*equations*>
else
  <*equations*>
end if;

```
model Diode "Ideal diode"
  extends TwoPin;
  Real s;
  Boolean off;
equation
  off = s < 0;
  if off then
    v=s
  else
    v=0;
  end if;
  i = if off then 0 else s;
end Diode;
```

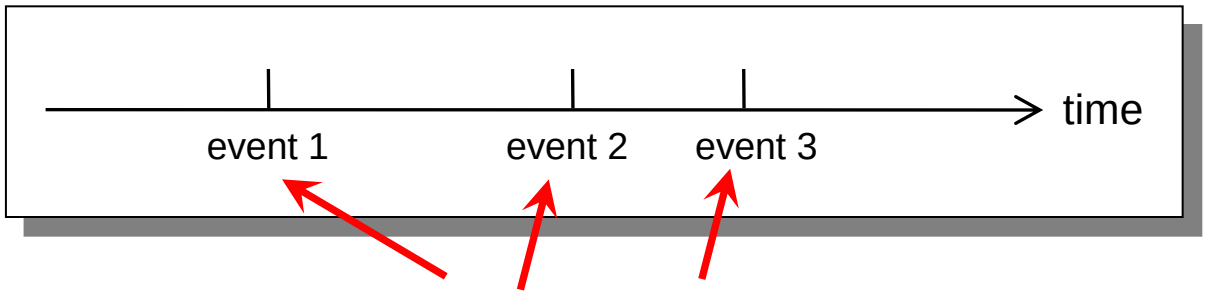


Copyright © Open Source Modelica Consortium

Event Creation – when

when-equations

when <*conditions*> **then**
  <*equations*>
end when;



Equations only active at event times

Time event

```
when time >= 10.0 then
  ...
end when;
```

Only dependent on time, can be scheduled in advance

State event

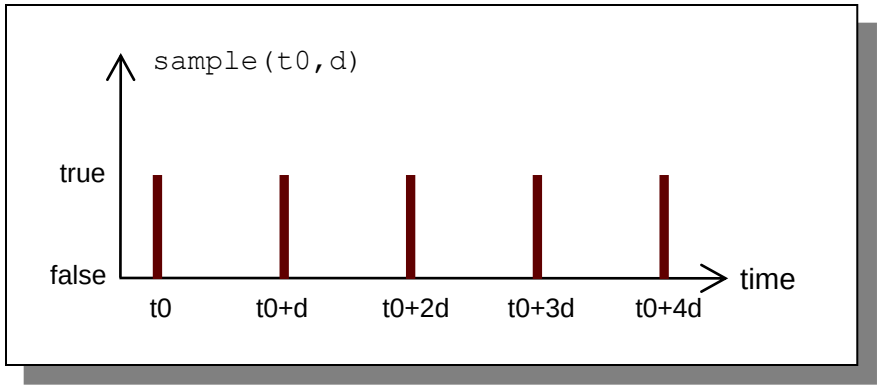
```
when sin(x) > 0.5 then
  ...
end when;
```

Related to a state. Check for zero-crossing

Copyright © Open Source Modelica Consortium

Generating Repeated Events

The call `sample(t0,d)` returns true and triggers events at times `t0+i*d`, where `i=0,1,...`

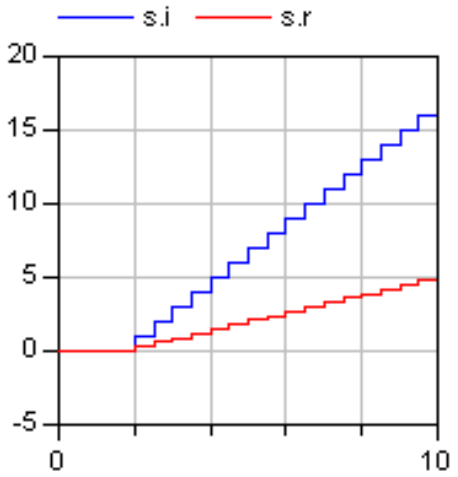


Variables need to be discrete

```
model SamplingClock
  Integer i;
  discrete Real r;
equation
  when sample(2,0.5) then
    i = pre(i)+1;
    r = pre(r)+0.3;
  end when;
end SamplingClock;
```

Creates an event after 2 s, then each 0.5 s

pre(...) takes the previous value before the event.

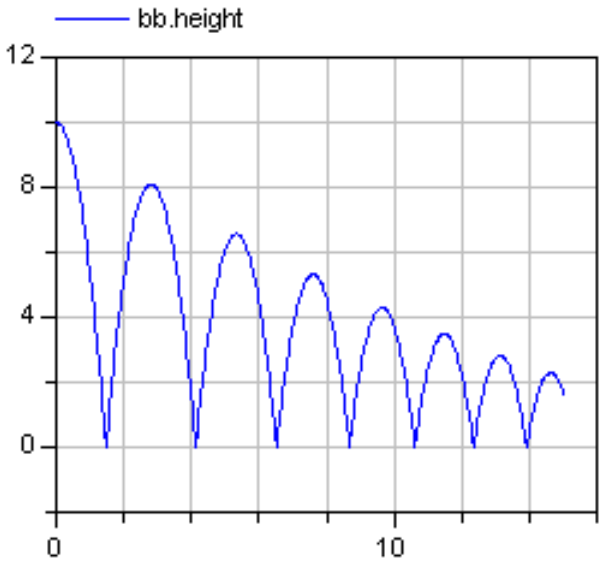


Copyright © Open Source Modelica Consortium

Reinit - Discontinuous Changes

The value of a *continuous-time* state variable can be instantaneously changed by a `reinit`-equation within a `when`-equation

```
model BouncingBall "the bouncing ball model"
  parameter Real g=9.81;   //gravitational acc.
  parameter Real c=0.90;   //elasticity constant
  Real height(start=10),velocity(start=0);
equation
  der(height) = velocity;
  der(velocity)=-g;
  when height<0 then
    reinit(velocity, -c*velocity);
  end when;
end BouncingBall;
```



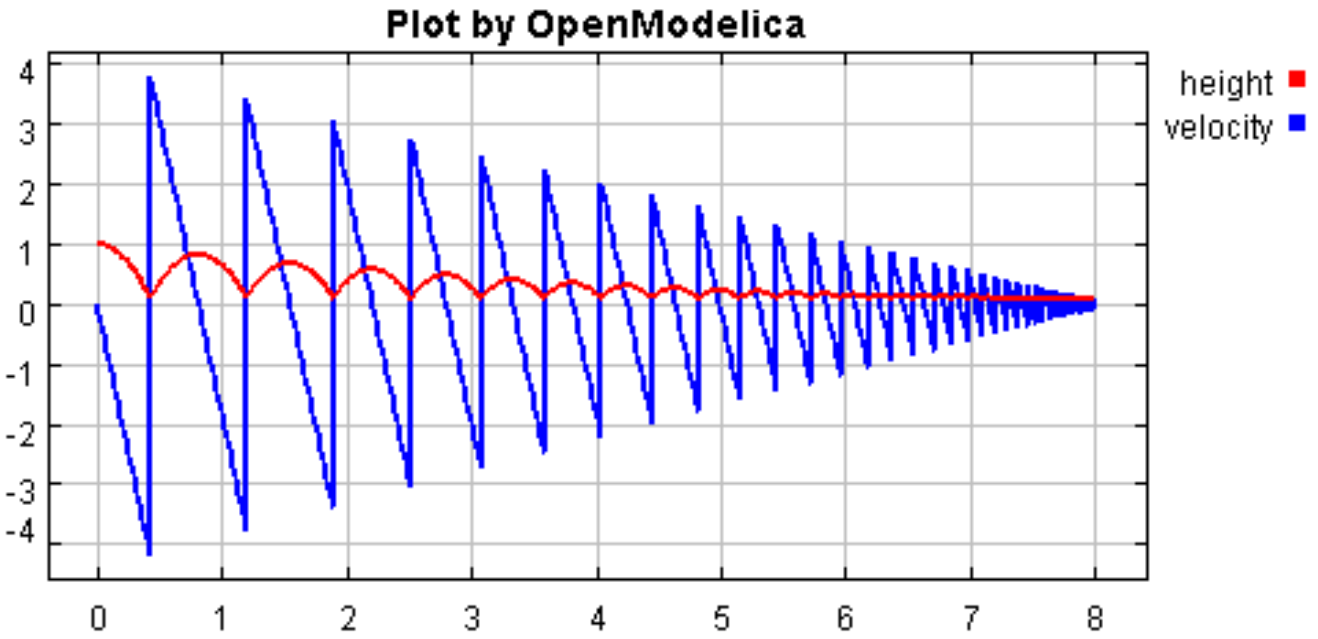
Initial conditions

Reinit "assigns" continuous-time variable `velocity` a new value

Copyright © Open Source Modelica Consortium

Exercise 2.6 – BouncingBall

- Locate the BouncingBall model in one of the hybrid modeling sections of DrModelica (the When-Equations link in Section 2.9), run it, change it slightly, and re-run it.



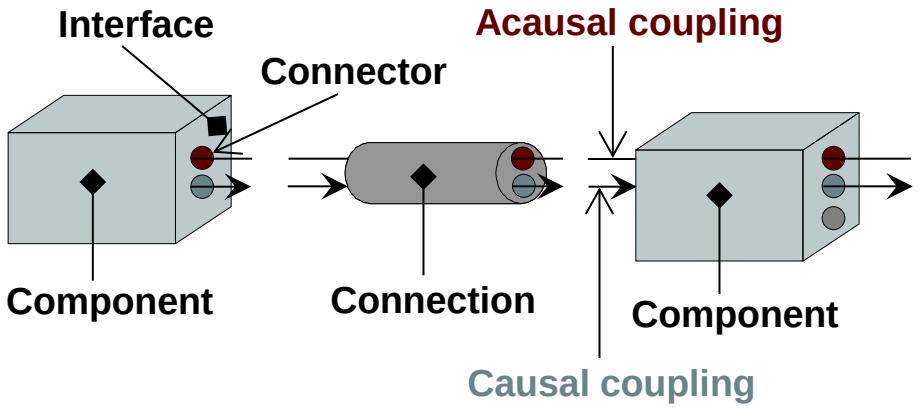
Copyright © Open Source Modelica Consortium

Part IV

Components, Connectors and Connections – Modelica Libraries and Graphical Modeling

Copyright © Open Source Modelica Consortium

Software Component Model



A component class should be defined *independently of the environment,* very essential for *reusability*

A component may internally consist of other components, i.e. *hierarchical* modeling

Complex systems usually consist of large numbers of *connected* components

Copyright © Open Source Modelica Consortium

Connectors and Connector Classes

Connectors are instances of *connector classes*

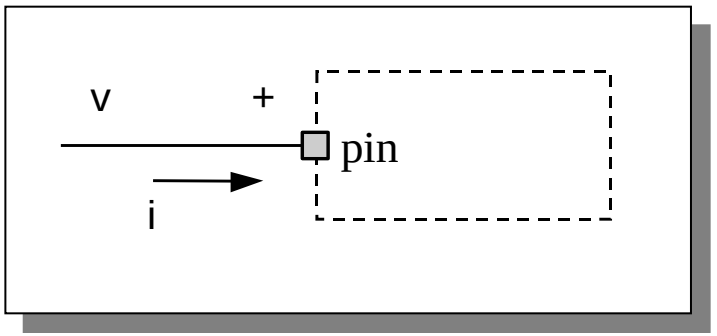
electrical connector

connector class

keyword `flow`
indicates that currents
of connected pins
sum to zero.

an instance `pin`
of class `Pin`

```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;

Pin pin;
```

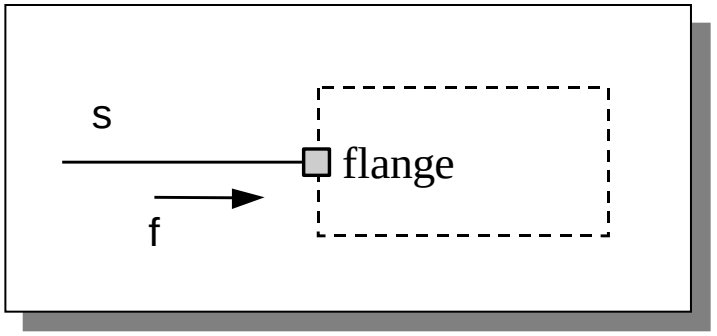


mechanical connector

connector class

an instance `flange`
of class `Flange`

```
connector Flange
  Position     s;
  flow Force   f;
end Flange;

Flange flange;
```



Copyright © Open Source Modelica Consortium

The `flow` prefix

Two kinds of variables in connectors:

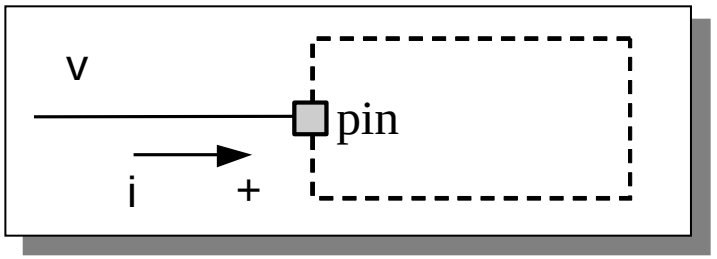
- *Potential variables* *potential* or energy level
- *Flow variables* represent some kind of flow

Coupling

- *Equality coupling*, for non-`flow` variables
- *Sum-to-zero coupling*, for `flow` variables

The value of a `flow` variable is *positive* when the current or the flow is *into* the component

positive flow direction:



Copyright © Open Source Modelica Consortium

Physical Connector

- Classes Based on Energy Flow

Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	`Electrical.Analog`
Translational	Position	Force	Linear momentum	`Mechanical.Translational`
Rotational	Angle	Torque	Angular momentum	`Mechanical.Rotational`
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux	
Hydraulic	Pressure	Volume flow	Volume	`OpenHydraulics`
Heat	Temperature	Heat flow	Heat	`HeatFlow1D`
Chemical	Chemical potential	Particle flow	Particles	Under construction
Pneumatic	Pressure	Mass flow	Air	`PneuLibLight`

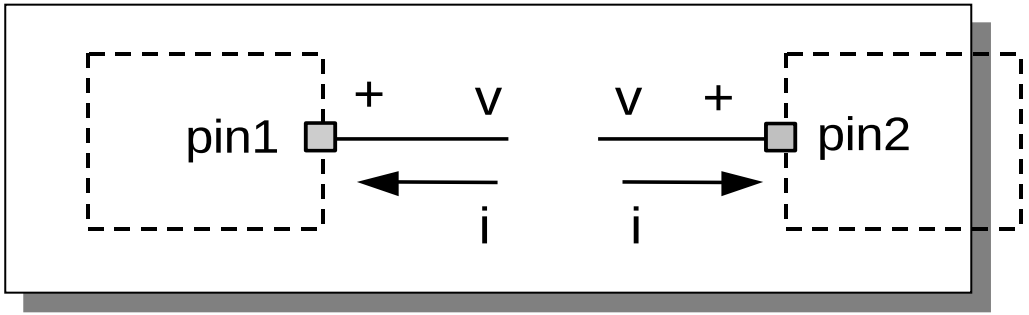
Copyright © Open Source Modelica Consortium

connect-equations

Connections between connectors are realized as *equations* in Modelica

```
connect(connector1,connector2)
```

The two arguments of a `connect`-equation must be references to *connectors*, either to be declared directly *within* the *same class* or be *members* of one of the declared variables in that class




```
Pin pin1,pin2;
//A connect equation
//in Modelica:
connect(pin1,pin2);
```

Corresponds to

```
pin1.v = pin2.v;
pin1.i + pin2.i =0;
```


Copyright © Open Source Modelica Consortium

Connection Equations

```
Pin pin1,pin2;
//A connect equation
//in Modelica
connect(pin1,pin2);
```

Corresponds to

```
pin1.v = pin2.v;
pin1.i + pin2.i =0;
```

Multiple connections are possible:

```
connect(pin1,pin2); connect(pin1,pin3); ... connect(pin1,pinN);
```

Each primitive connection set of potential variables is used to generate equations of the form:

$$v_1 = v_2 = v_3 = \ldots v_n$$

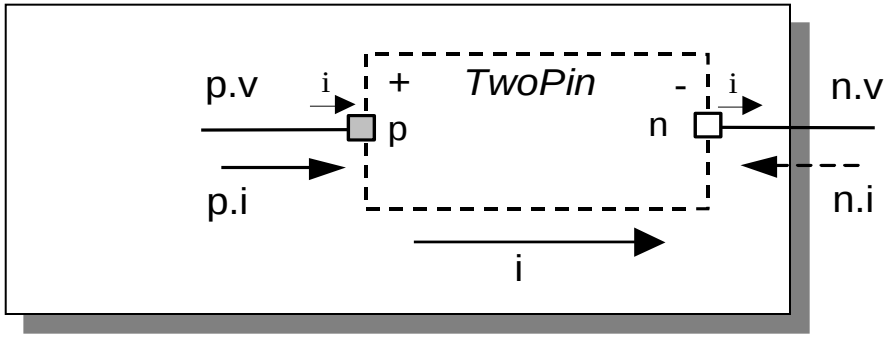
Each primitive connection set of flow variables is used to generate *sum-to-zero* equations of the form:

$$i_1 + i_2 + \ldots(-i_k) + \ldots i_n = 0$$

Copyright © Open Source Modelica Consortium

Common Component Structure

The base class `TwoPin` has two connectors `p` and `n` for positive and negative pins respectively



partial class (cannot be instantiated)

positive pin

negative pin

```
partial model TwoPin
  Voltage     v
  Current     i
  Pin p;
  Pin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
// TwoPin is same as OnePort in
// Modelica.Electrical.Analog.Interfaces
```

```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;
```

electrical connector class

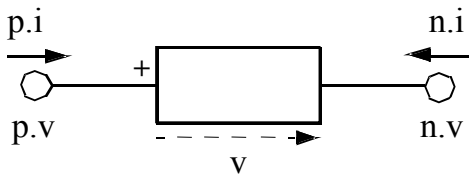
Copyright © Open Source Modelica Consortium

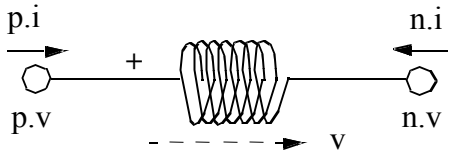
Electrical Components

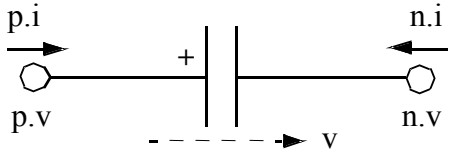
```
model Resistor "Ideal electrical resistor"
  extends TwoPin;
  parameter Real R;
equation
  R*i = v;
end Resistor;
```

```
model Inductor "Ideal electrical inductor"
  extends TwoPin;
  parameter Real L "Inductance";
equation
  L*der(i) = v;
end Inductor;
```

```
model Capacitor "Ideal electrical capacitor"
  extends TwoPin;
  parameter Real C ;
equation
  i=C*der(v);
end Capacitor;
```



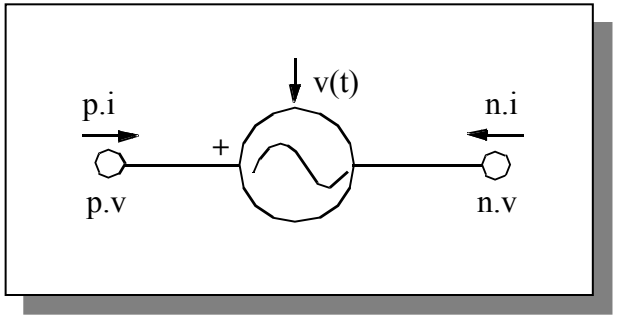




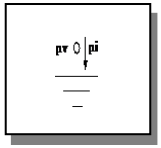
Copyright © Open Source Modelica Consortium

Electrical Components cont'

```
model Source
   extends TwoPin;
   parameter Real A,w;
equation
   v = A*sin(w*time);
end Resistor;
```

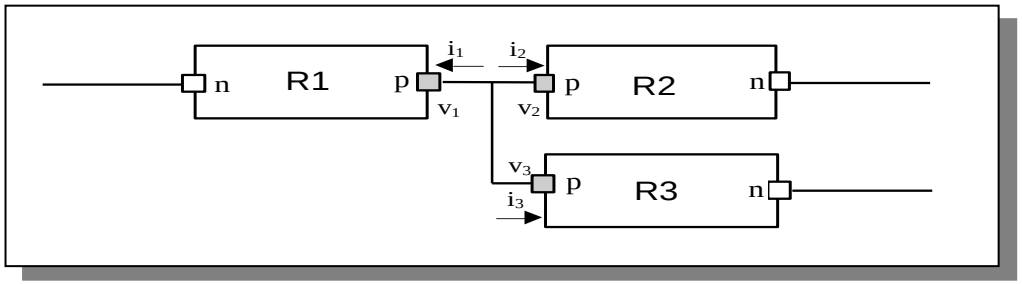



```
model Ground
   Pin p;
equation
   p.v = 0;
end Ground;
```



Copyright © Open Source Modelica Consortium

Resistor Circuit




```
model ResistorCircuit
  Resistor R1(R=100);
  Resistor R2(R=200);
  Resistor R3(R=300);
equation
  connect(R1.p, R2.p);
  connect(R1.p, R3.p);
end ResistorCircuit;
```

Corresponds to

```
R1.p.v = R2.p.v;
R1.p.v = R3.p.v;
R1.p.i + R2.p.i + R3.p.i = 0;
```


Copyright © Open Source Modelica Consortium

Modelica Standard Library - Graphical Modeling

- *Modelica Standard Library* (called Modelica) is a standardized predefined package developed by Modelica Association

- It can be used freely for both commercial and noncommercial purposes under the conditions of *The Modelica License*.

- Modelica libraries are available online including documentation and source code from http://www.modelica.org/library/library.html

Copyright © Open Source Modelica Consortium

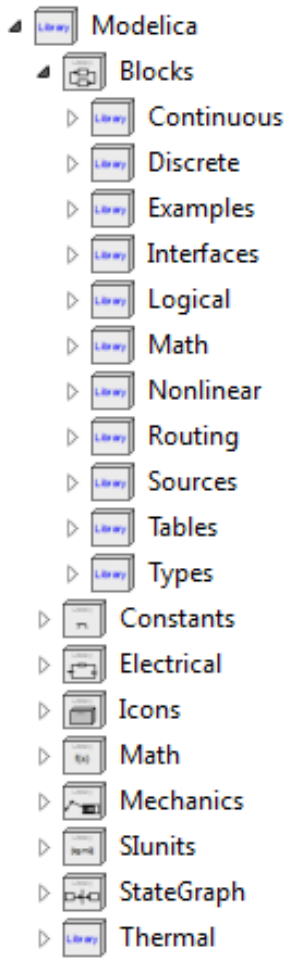
Modelica Standard Library cont'

The Modelica Standard Library contains components from various application areas, including the following sublibraries:

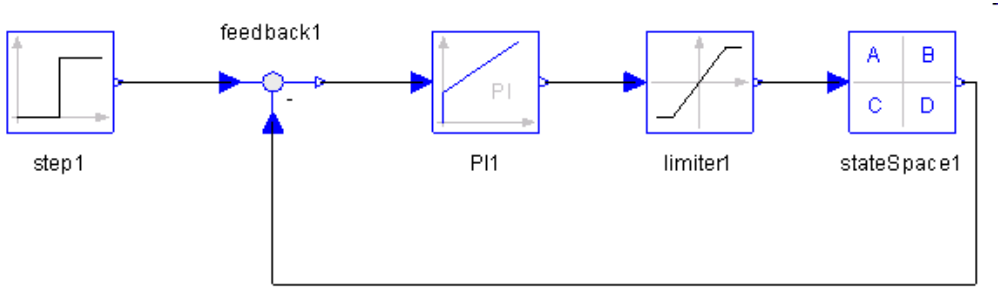
- Blocks — Library for basic input/output control blocks
- Constants — Mathematical constants and constants of nature
- Electrical — Library for electrical models
- Icons — Icon definitions
- Fluid — 1-dim Flow in networks of vessels, pipes, fluid machines, valves, etc.
- Math — Mathematical functions
- Magnetic — Magnetic.Fluxtubes – for magnetic applications
- Mechanics — Library for mechanical systems
- Media — Media models for liquids and gases
- SIunits — Type definitions based on SI units according to ISO 31-1992
- Stategraph — Hierarchical state machines (analogous to Statecharts)
- Thermal — Components for thermal systems
- Utilities — Utility functions especially for scripting

Copyright © Open Source Modelica Consortium

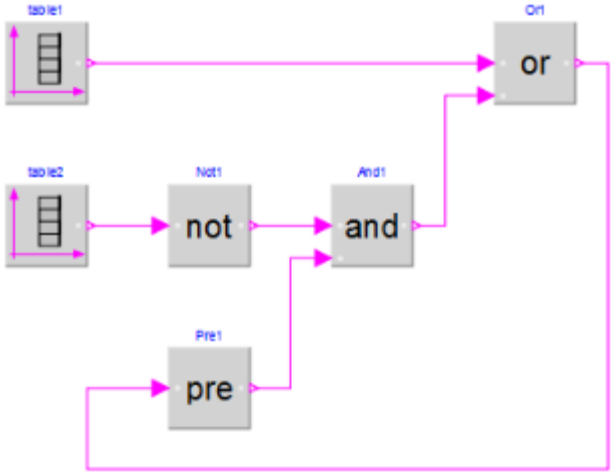
Modelica.Blocks



Continuous, discrete, and logical input/output blocks to build block diagrams.



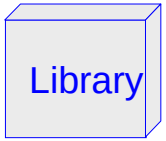
Examples:



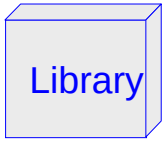
Copyright © Open Source Modelica Consortium

Modelica.Electrical

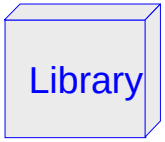
Electrical components for building analog, digital, and multiphase circuits



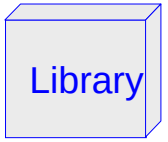
Analog



Digital

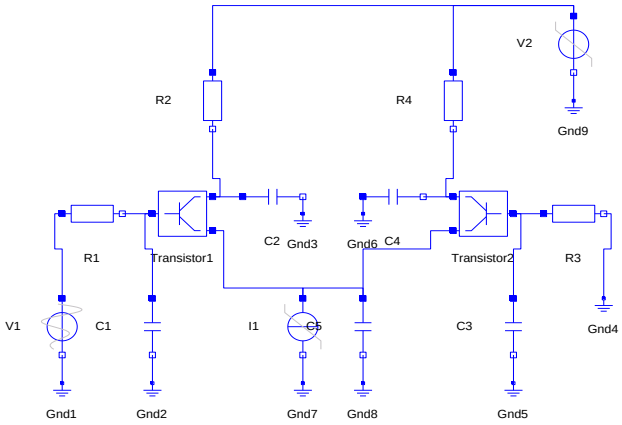


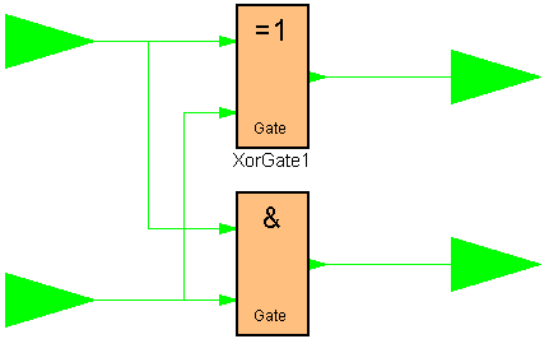
Machines

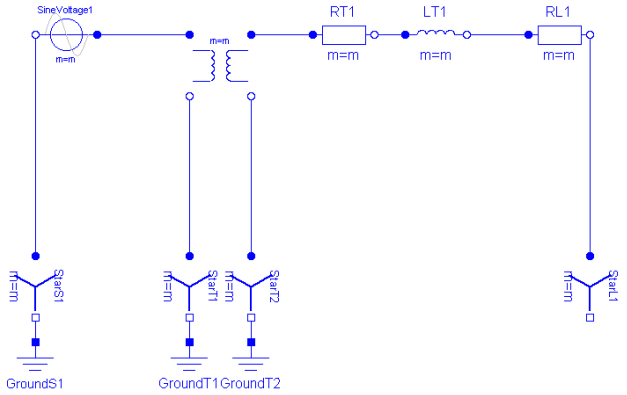


MultiPhase

Examples:







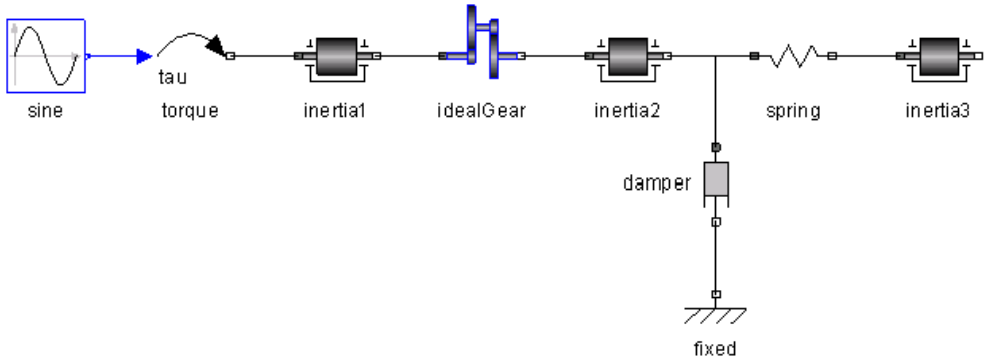
Copyright © Open Source Modelica Consortium

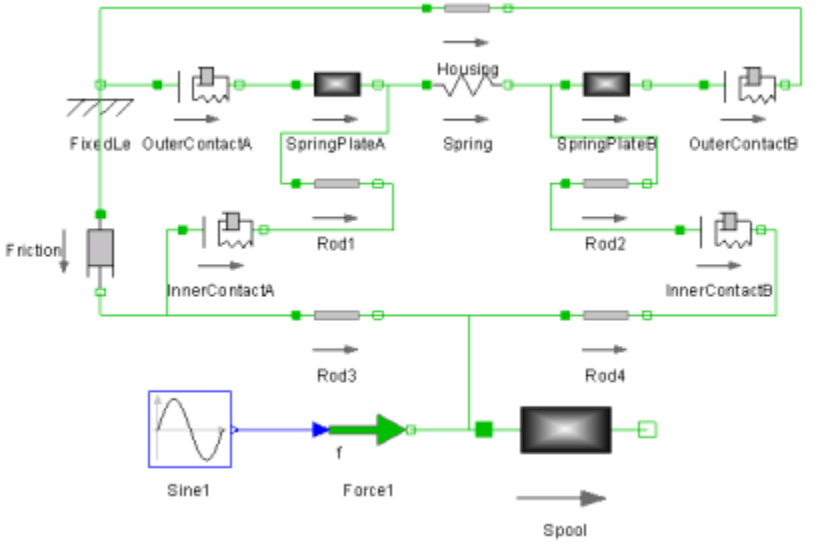
Modelica.Mechanics

Package containing components for mechanical systems

Subpackages:

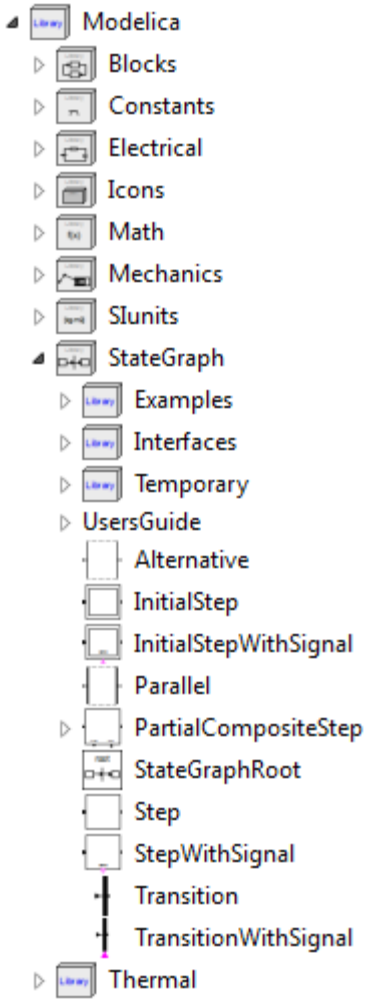
- Rotational — 1-dimensional rotational mechanical components
- Translational — 1-dimensional translational mechanical components
- MultiBody — 3-dimensional mechanical components



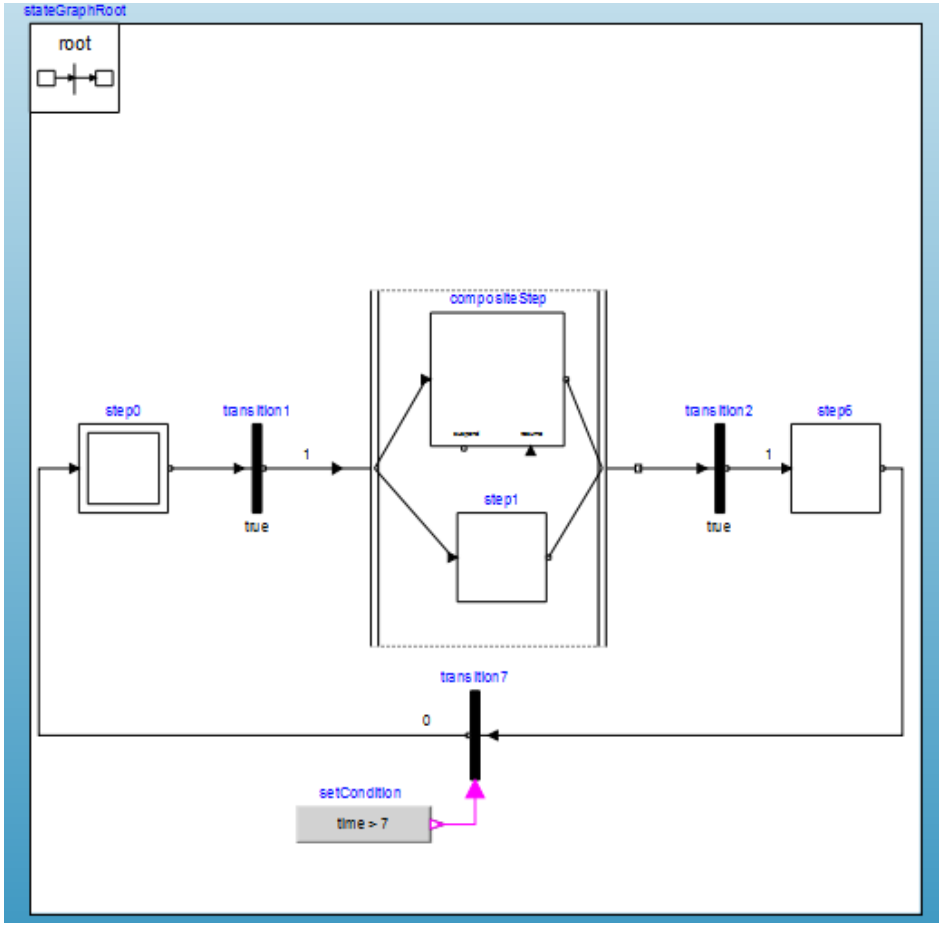


Copyright © Open Source Modelica Consortium

Modelica.Stategraph



Hierarchical state machines (similar to Statecharts)



Copyright © Open Source Modelica Consortium

Other Free Libraries

- WasteWater — Wastewater treatment plants, 2003
- ATPlus — Building simulation and control (fuzzy control included), 2005
- MotorCycleDymanics — Dynamics and control of motorcycles, 2009
- NeuralNetwork — Neural network mathematical models, 2006
- VehicleDynamics — Dynamics of vehicle chassis (obsolete), 2003
- SPICElib — Some capabilities of electric circuit simulator PSPICE, 2003
- SystemDynamics — System dynamics modeling a la J. Forrester, 2007
- BondLib — Bond graph modeling of physical systems, 2007
- MultiBondLib — Multi bond graph modeling of physical systems, 2007
- ModelicaDEVS — DEVS discrete event modeling, 2006
- ExtendedPetriNets — Petri net modeling, 2002
- External.Media Library — External fluid property computation, 2008
- VirtualLabBuilder — Implementation of virtual labs, 2007
- SPOT — Power systems in transient and steady-state mode, 2007
- ...

Copyright © Open Source Modelica Consortium

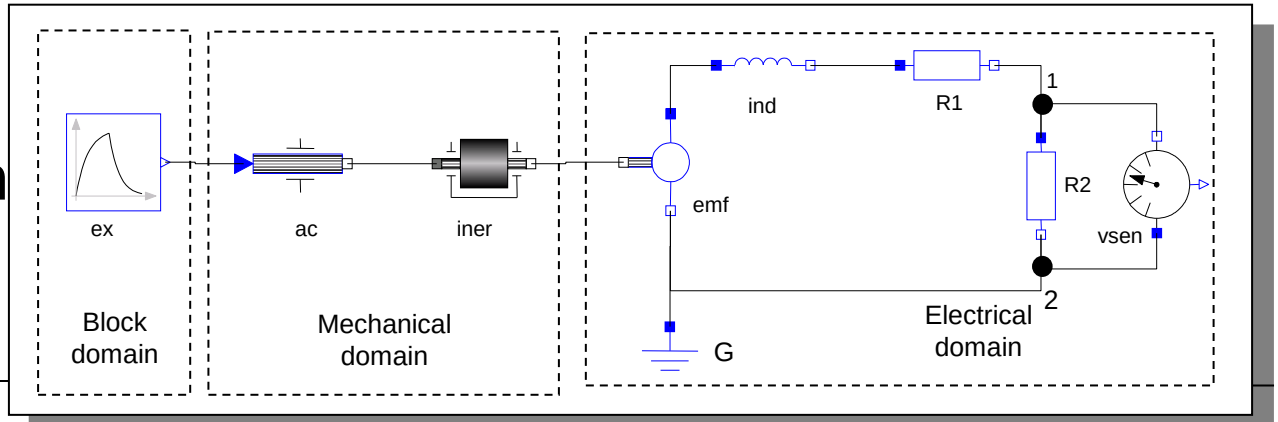
Some Commercial Libraries

- Powertrain
- SmartElectricDrives
- VehicleDynamics
- AirConditioning
- HyLib
- PneuLib
- CombiPlant
- HydroPlant
- …

Copyright © Open Source Modelica Consortium

Connecting Components from Multiple Domains

- **Block domain**
- **Mechanical domain**
- **Electrical domain**



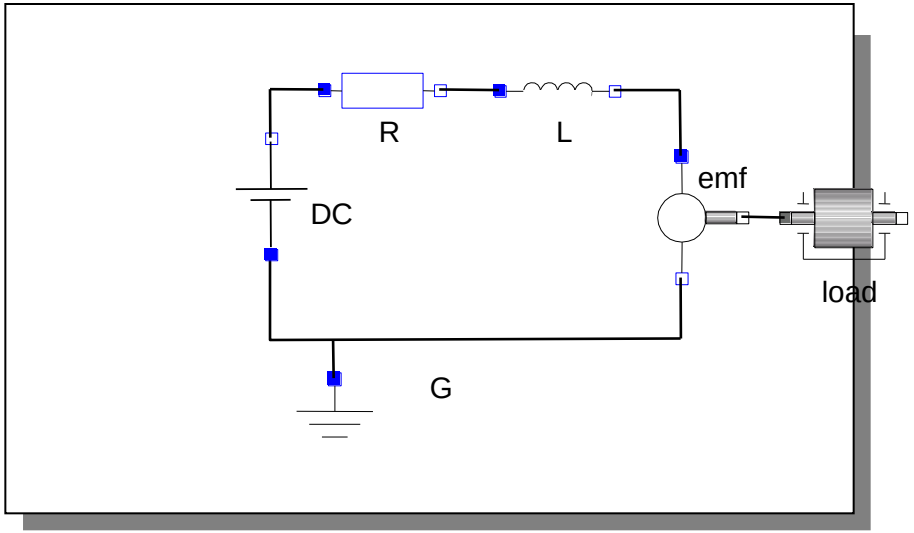

```
model Generator
  Modelica.Mechanics.Rotational.Accelerate ac;
  Modelica.Mechanics.Rotational.Inertia iner;
  Modelica.Electrical.Analog.Basic.EMF emf(k=-1);
  Modelica.Electrical.Analog.Basic.Inductor ind(L=0.1);
  Modelica.Electrical.Analog.Basic.Resistor R1,R2;
  Modelica.Electrical.Analog.Basic.Ground G;
  Modelica.Electrical.Analog.Sensors.VoltageSensor vsens;
  Modelica.Blocks.Sources.Exponentials ex(riseTime={2},riseTimeConst={1});
equation
  connect(ac.flange_b, iner.flange_a); connect(iner.flange_b, emf.flange_b);
  connect(emf.p, ind.p); connect(ind.n, R1.p); connect(emf.n, G.p);
  connect(emf.n, R2.n); connect(R1.n, R2.p); connect(R2.p, vsens.n);
  connect(R2.n, vsens.p); connect(ex.outPort, ac.inPort);
end Generator;
```


Copyright © Open Source Modelica Consortium

DCMotor Model Multi-Domain (Electro-Mechanical)

A DC motor can be thought of as an electrical circuit which also contains an electromechanical component.

```
model DCMotor
   Resistor R(R=100);
   Inductor L(L=100);
   VsourceDC DC(f=10);
   Ground G;
   EMF emf(k=10,J=10, b=2);
   Inertia load;
equation
   connect(DC.p,R.n);
   connect(R.p,L.n);
   connect(L.p, emf.n);
   connect(emf.p, DC.n);
   connect(DC.n,G.p);
   connect(emf.flange,load.flange);
end DCMotor;
```



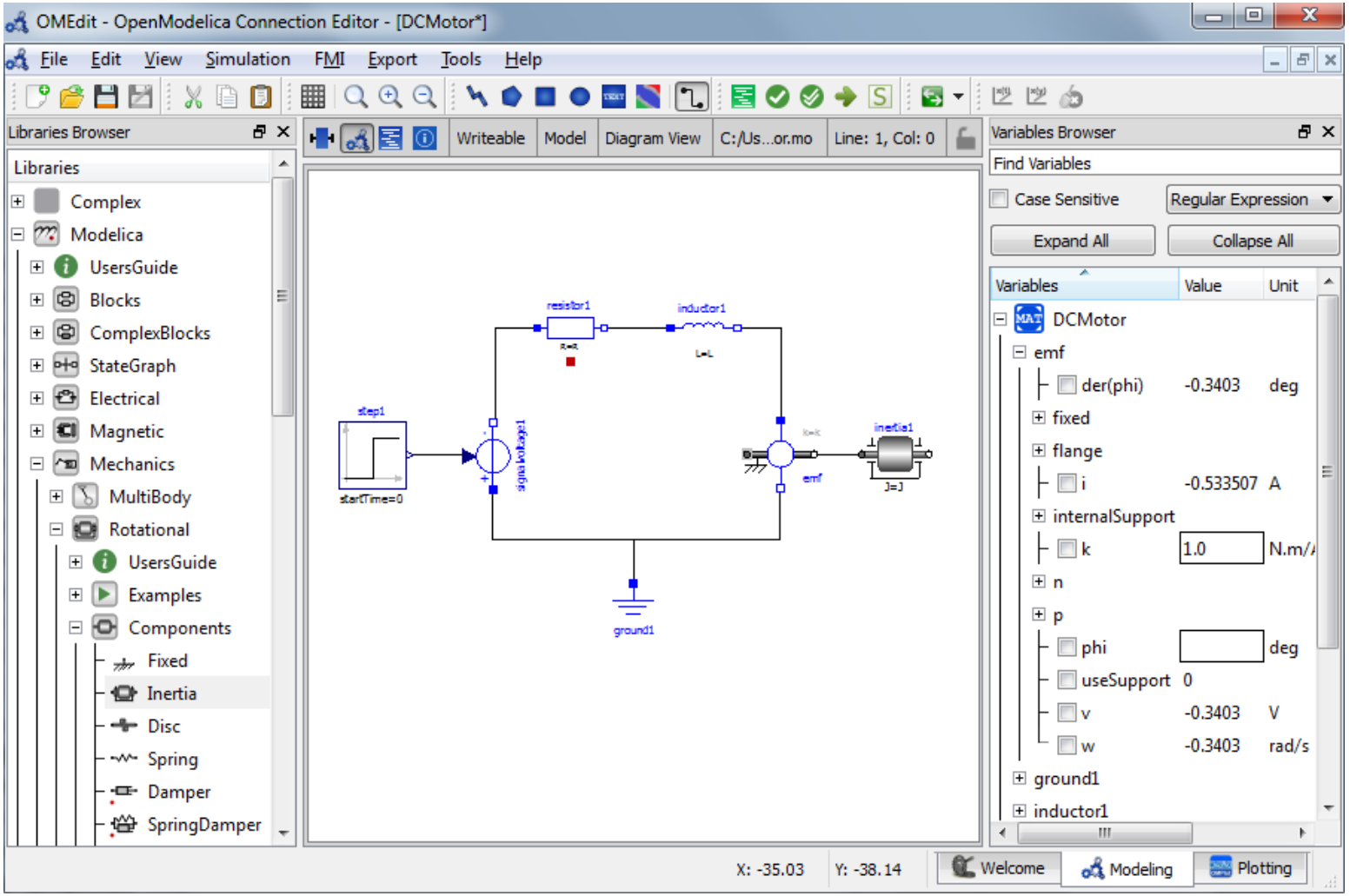
Copyright © Open Source Modelica Consortium

Exercises Part IV
Graphical Modeling Exercises

using
OpenModelica

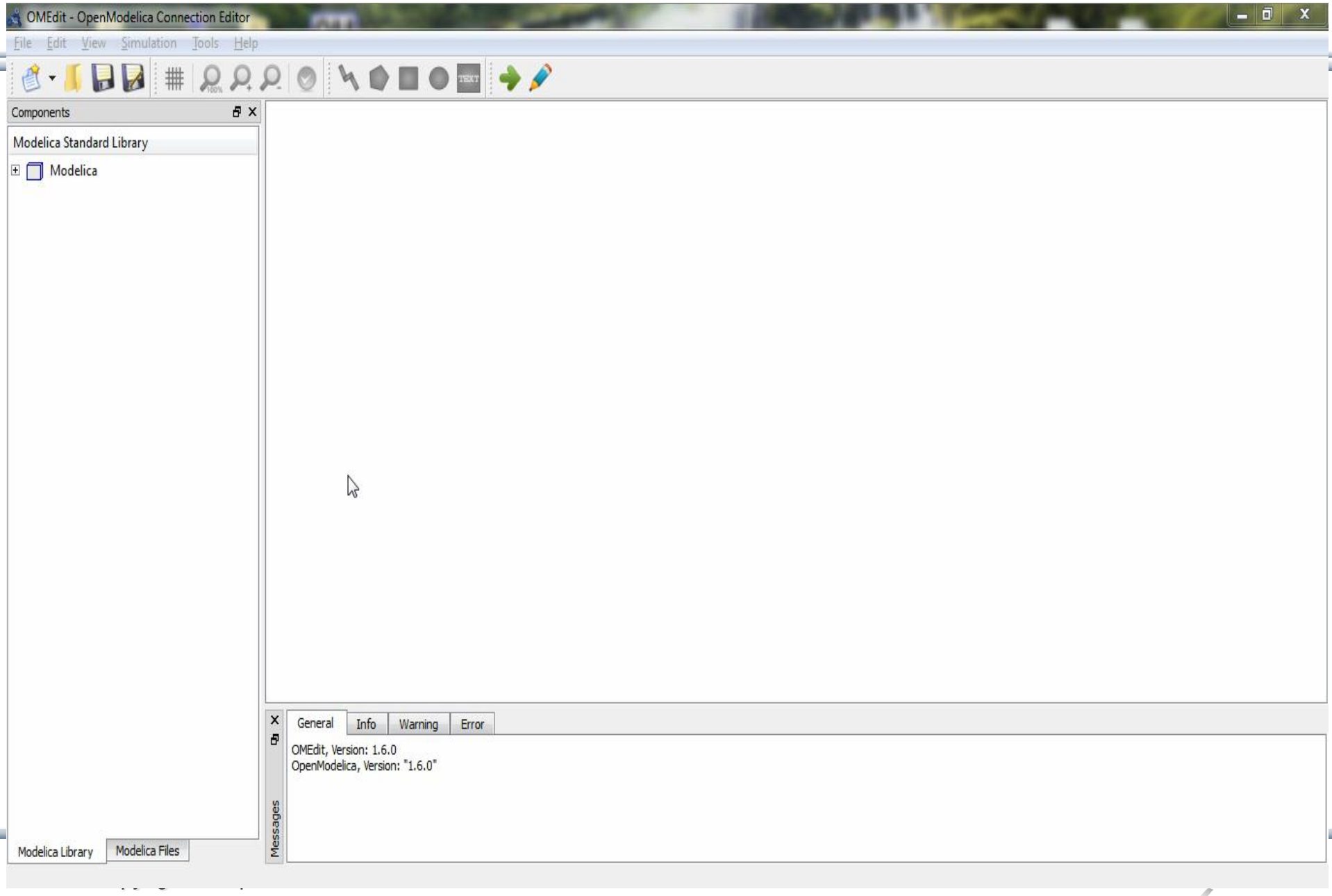
Copyright © Open Source Modelica Consortium

Graphical Modeling - Using Drag and Drop Composition



Copyright © Open Source Modelica Consortium

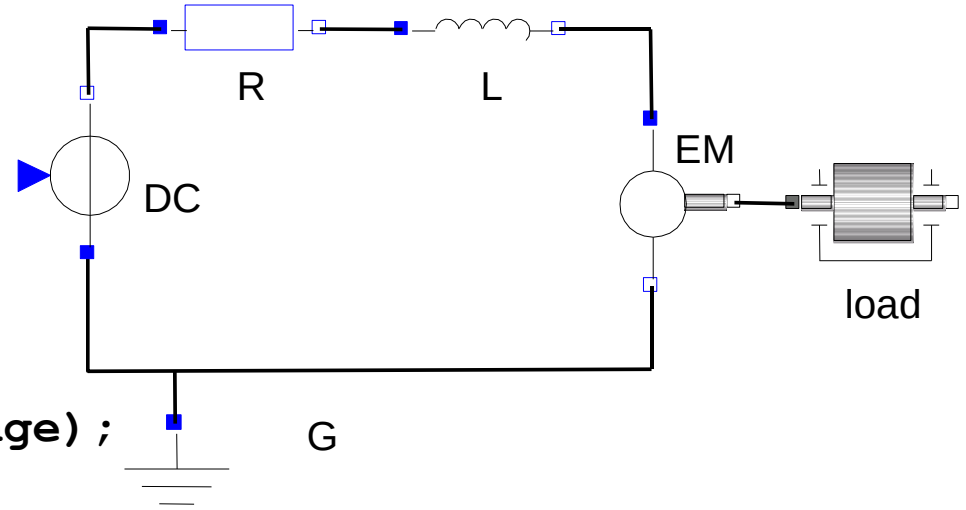
Graphical Modeling Animation – DCMotor



Multi-Domain (Electro-Mechanical) Modelica Model

- A DC motor can be thought of as an electrical circuit which also contains an electromechanical component

```
model DCMotor
   Resistor R(R=100);
   Inductor L(L=100);
   VsourceDC DC(f=10);
   Ground G;
   ElectroMechanicalElement EM(k=10,J=10, b=2);
   Inertia load;
equation
   connect(DC.p,R.n);
   connect(R.p,L.n);
   connect(L.p, EM.n);
   connect(EM.p, DC.n);
   connect(DC.n,G.p);
   connect(EM.flange,load.flange);
end DCMotor
```



Copyright © Open Source Modelica Consortium

Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

```
0 == DC.p.i + R.n.i
DC.p.v == R.n.v

0 == R.p.i + L.n.i
R.p.v == L.n.v

0 == L.p.i + EM.n.i
L.p.v == EM.n.v

0 == EM.p.i + DC.n.i
EM.p.v == DC.n.v

0 == DC.n.i + G.p.i
DC.n.v == G.p.v

EM.u == EM.p.v - EM.n.v
0 == EM.p.i + EM.n.i
EM.i == EM.p.i
EM.u == EM.k * EM.ω
EM.i == EM.M / EM.k
EM.J * EM.ω == EM.M - EM.b * EM.ω

DC.u == DC.p.v - DC.n.v
0 == DC.p.i + DC.n.i
DC.i == DC.p.i
DC.u == DC.Amp * Sin[2 π DC.f * t]

R.u == R.p.v - R.n.v
0 == R.p.i + R.n.i
R.i == R.p.i
R.u == R.R * R.i

L.u == L.p.v - L.n.v
0 == L.p.i + L.n.i
L.i == L.p.i
L.u == L.L * L.i'
```

(`load` component not included)

Automatic transformation to ODE or DAE for simulation:

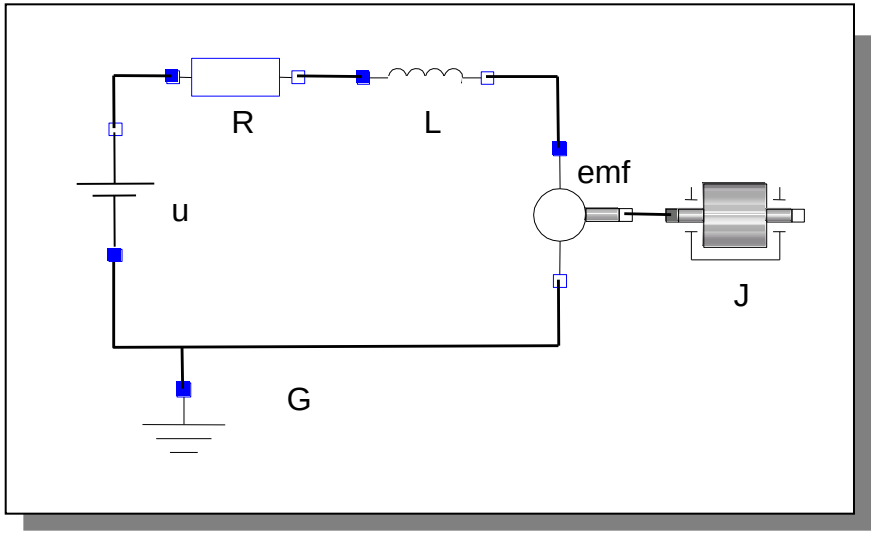
$$\frac{dx}{dt} == f[x, u, t] \qquad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

Copyright © Open Source Modelica Consortium

Exercise 3.1

- Draw the `DCMotor` model using the graphic connection editor using models from the following Modelica libraries:
  `Mechanics.Rotational.Components`,
  `Electrical.Analog.Basic`,
  `Electrical.Analog.Sources`

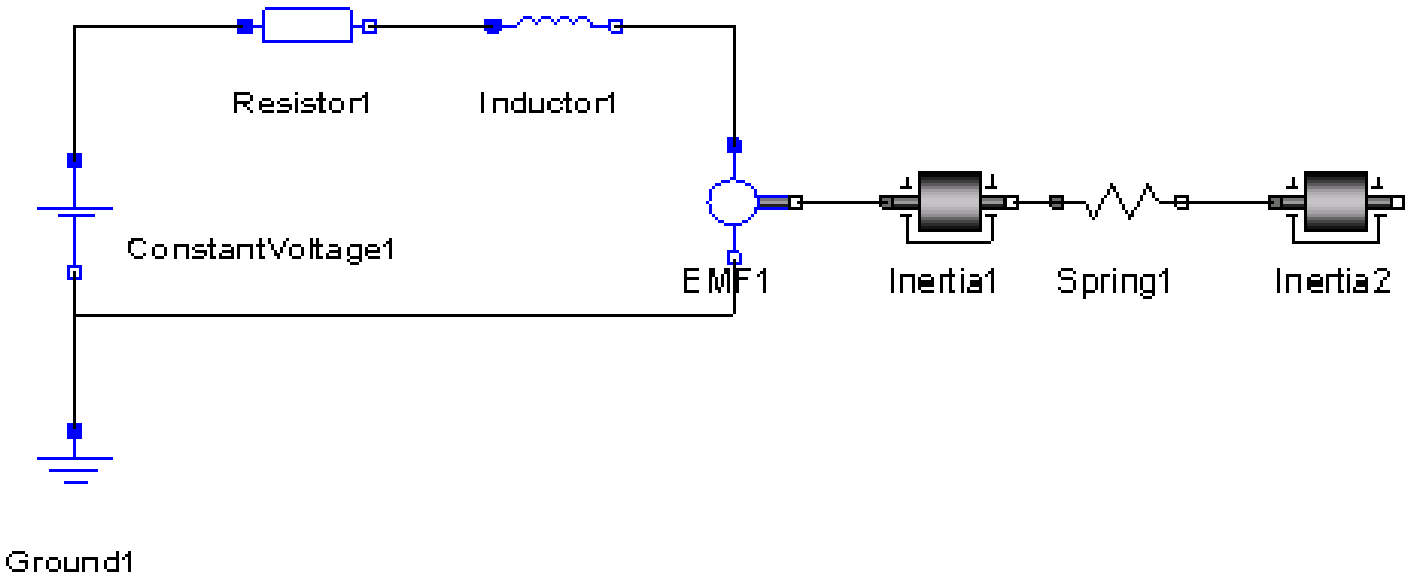
- Simulate it for 15s and plot the variables for the outgoing rotational speed on the inertia axis and the voltage on the voltage source (denoted u in the figure) in the same plot.



Copyright © Open Source Modelica Consortium

Exercise 3.2

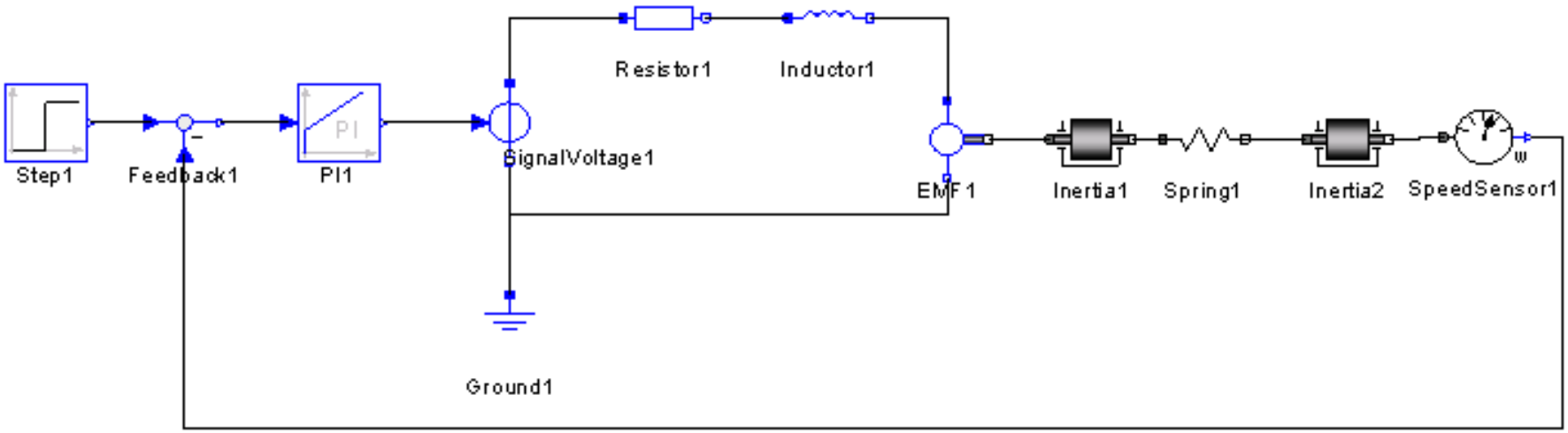
- If there is enough time: Add a torsional spring to the outgoing shaft and another inertia element. Simulate again and see the results. Adjust some parameters to make a rather stiff spring.



Copyright © Open Source Modelica Consortium

Exercise 3.3

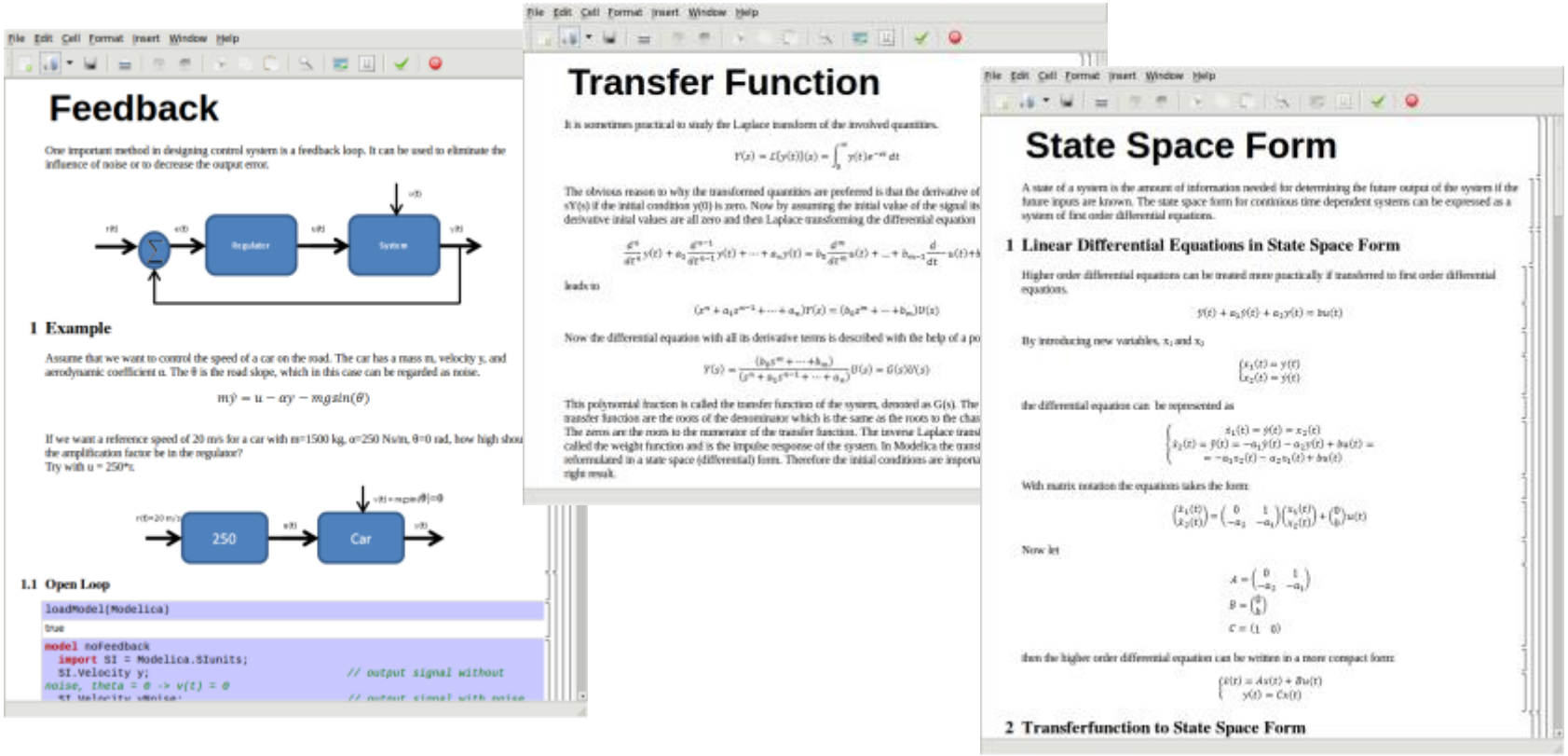
- If there is enough time: Add a PI controller to the system and try to control the rotational speed of the outgoing shaft. Verify the result using a step signal for input. Tune the PI controller by changing its parameters in OMEdit.



Copyright © Open Source Modelica Consortium

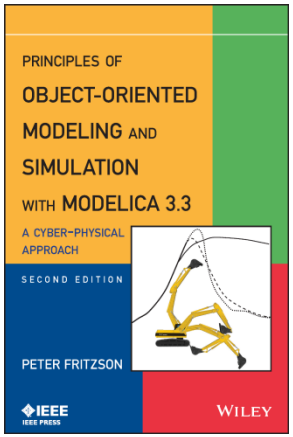
Exercise 3.4 – DrControl

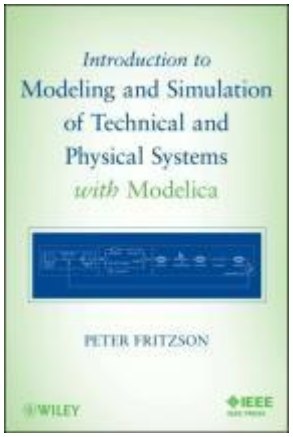
- If there is enough time: Open the DrControl electronic book about control theory with Modelica and do some exercises.
  - **Open File**: C:OpenModelica1.6.0\share\omnotebook\drcontrol\DrControl.onb



Copyright © Open Source Modelica Consortium

Learn more…





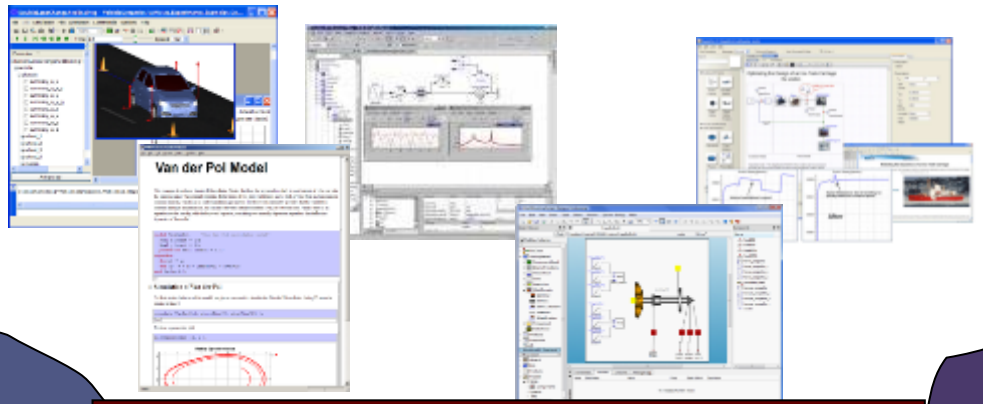
- OpenModelica
  - www.openmodelica.org
- Modelica Association
  - www.modelica.org
- Books
  - Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach, Peter Fritzson
  - Modeling and Simulation of Technical and Physical Systems with Modelica. Peter Fritzson. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-111801068X.html
  - Introduction to Modelica, Michael Tiller

Copyright © Open Source Modelica Consortium

Summary

Multi-Domain Modeling

Visual Acausal Component Modeling



Typed Declarative Textual Language

Thanks for listening!

Hybrid Modeling

Copyright © Open Source Modelica Consortium